Studies on Designing Distributed and Cooperative Systems for Solving Constraint Satisfaction Problems of Container Loading

Yuan Liu

Acknowledgments

First and foremost, I owe many thanks to my supervisor, Professor Tetsuo Sawaragi, for his excellent guidance, patient support. More importantly, his rigorous scholarship influenced me greatly.

I also wish give special thanks to Ph.D. Yajie Tian because she also has given me much excellent guidance, patient guidance and encourage.

I would like to thank all members in Sawaragi laboratory for helping me solving many troubles in life.

Finally, I would like to thank my family because they were always with me and gave me boundless love and understanding.

Abstract

In the real world, many difficult problems exist in manufacturing and engineering, including logistics problems and scheduling problems. They are defined as *constraint satisfaction problem*s (CSPs) in the *operation research* and *artificial intelligence* fields. Even suboptimal solutions to a CSP are difficult to obtain, not only because of multiple restrictions, but also because of multiple objectives.

Many optimization algorithms, multi-agent-based algorithms or both are available for solving different kinds of CSPs. But algorithms proposed so far cannot solve CSPs in the real world because involved restrictions are sometimes self-contradictory and difficult to satisfy simultaneously. Beyond that, there is never enough computation time to get an optimal solution. This research focuses on solving these kinds of CSPs using distributed and cooperative approaches and discusses the general issue of the container loading problem (CLP) as an example.

When human experts deal with complicated restrictions, they have the know-how to classify restrictions and relax some of them with the goal of finding a *satisficing* solution rather than an optimal solution. To simulate this kind of human expert flexibility, "Theory of Constraints-based Management" (TOCM) is proposed. In TOCM, restrictions are divided into hard and soft, according to the situations on site and user requests. Hard restrictions must be satisfied even though solution effectiveness is poor and soft restrictions may be relaxed according to specific situations. Further, TOCM is based on the distributed Multi-agent System and Theory of Constraints. A particular target of TOCM focuses on improving the constrained agent by striving for integrated improvement of its restrictions and making agents negotiate cooperatively. The goal is to arrive at a final solution that yields the biggest profit for a system.

Because TOCM cannot guarantee that every soft restriction of an agent is well-satisfied, an approach called "Three-zone Buffer Management" (TZBM) is introduced and improved. In TZBM, each soft restriction corresponds to a relaxable variable and the value of the variable reflects the extent to which the soft restriction is violated. Each agent is divided into one of three zones (Green-zone, Yellow-zone and Red-zone) according to the worst satisfied situation of its own soft restrictions. The constrained agent in the Red-zone is improved by striving for trade-offs of its soft restrictions and making agents negotiate cooperatively. When a system runs

into a deadlock that prevents the Red-zone from being emptied, soft restrictions are relaxed and agents are re-divided into three zones. The result is that more opportunities are revealed for constrained agents in the Red-zone to be eliminated. A final satisficing solution is obtained until the Red-zone is emptied or relaxable variables reach their maximums. All soft restrictions are violated as seldom as possible.

Once a program is developed, it is seldom flexible or adaptable enough to deal with similar but differing problems because it has no ability to recognize new situations or adjust its existing problem-solving strategies. On the other hand, a human expert has this kind of experience to adapt to diversity. To use the human expert's ability and solve the "year-2007 problem" in Japan, an approach called "Interaction-based Knowledge Acquiring Framework" (IKAF) is proposed. The goal is to make a program with high flexibility and adaptability and for human expert knowledge to be retained in that expert's company. Because much of a human expert's knowledge is experience and tacit, it is difficult to implement it explicitly. In IKAF, a human expert agent is provided opportunities to use individual tacit skills in criticizing a result obtained using a conventional algorithm as well as to apply the heuristics gained to that point within a computer agent. From this performance, the computer agent detects knowledge fragments that are commonly lacking in existing knowledge-based systems, but are actually used in a human expert's performance. This detection is accomplished by applying the deductive learning approach of Explanation-based Learning. The distributed and cooperative procedures for transforming a human expert's tacit knowledge into explicit knowledge are presented, and examples of the knowledge acquisition are demonstrated in this research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Constraint Satisfaction Problem

In the real world, many difficult problems exist in manufacturing and engineering, including logistics problems and scheduling problems. A *constraint satisfaction problem* (CSP) is a mathematical problem in which the states or objects that satisfy a number of constraints or criteria must be found. A CSP is a subject of intense research in both the *artificial intelligence* and *operation research* fields.

CSPs in reality are very difficult problems. First, they are accompanied by multiple objectives and multiple restrictions that are difficult to define and evaluate because practical situations are always complicated and ambiguous. Specifically, multiple restrictions are usually self-contradictory and difficult to satisfy simultaneously. Second, they are usually NP-hard problems according to the computational complexity theory. Last and very important, traditional methods are always difficult to adapt to an even small change in an application because of its lack of the flexibility that comes natural to a human expert. For these reasons, it is usually difficult to obtain even a suboptimal solution to a CSP in the real world within a limited computation time.

For example, the container loading problem (CLP) is one kind of CSP in the real world [27]. The CLP holds an important place in logistics and has significant commercial applications.

A number of CLP categories have been considered [11]. One is based on the container quantity (single container [7] or multiple containers [30]). Another is based on the mix of cargo types to be loaded. In this respect, CLPs vary from a completely homogeneous problem, where cargoes have identical dimensions and orientations, to the extreme heterogeneous case, where many different sized items are present. The CLP between these two extremes (with relatively few cargo types) is often referred to as the weakly heterogeneous case [2]. The other separates CLPs into different loading patterns such as *layer-building* [12] [35], *tower-building* [11],

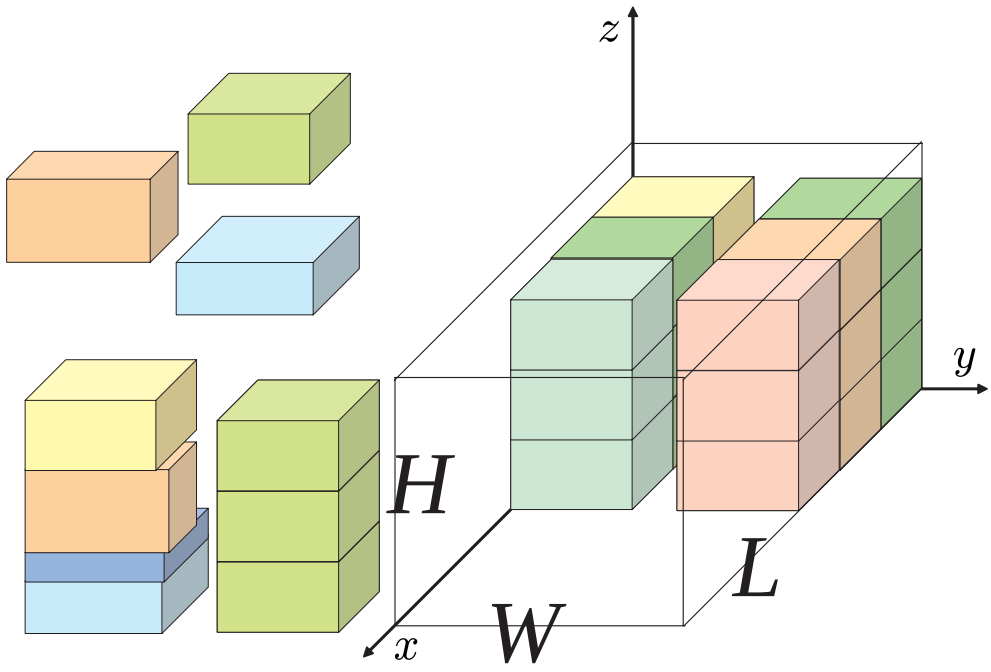


Figure 1.1: Making and loading modules of two-row pattern of CLP

and *block-building* [8] [20]. Loading patterns are decided by data properties and user requests. Generally, different mathematical models are proposed for solving different loading patterns.

In some cases (for instance, in large motor manufacturers), small parts are packed into boxes or placed on pallets, which are then loaded into containers; large parts are loaded into containers directly. Those to be loaded into containers directly are called cargoes in this research and are weakly heterogeneous. Because these cargoes are large and their lengths and widths are close to half the width of a container, a *two-row pattern* is used on site as shown in Figure 1.1. This pattern has two rows of modules lying against the two sidewalls of a container; each module is made by stacking cargoes vertically. The process for making modules is discussed in another paper [32] and omitted here. Only the process for loading modules, which is a two-dimensional allocation problem, is discussed in this research to explain our proposed approaches.

The *two-row pattern* of a CLP discussed in this research is a multi-destination delivery and multi-container problem. The difficulties to solve it are as follows:

1) Multiple restrictions are usually self-contradictory and difficult to satisfy simultaneously. First, volume and weight capacities of a container are limited. Second, a loading sequence for ordinal destinations has to be followed. And third, four kinds of desirable restrictions should be satisfied to the greatest extent possible. The

first is the gravity center, which needs to be close to the geometrical mid-point of the container floor. The second is the balance of row lengths. Good balance is possible by fixing the modules on the tails of two rows. The third is module crossing packing; similar gap widths between two rows are desirable. The fourth restriction is module stability. The cargo at the top of a module is best restricted by adjacent modules.

2) Beyond the basic objective of a minimum number of loaded containers, an additional objective to satisfy restrictions to the greatest extent possible must be taken into account.

3) The system goal is to achieve the optimum profit of all loaded containers rather than parts of them. That means every container should be loaded as efficiently and stably as possible.

4) The computation time to get a final solution is usually limited.

It is difficult to obtain an optimal solution because of many complicated restrictions and limited time.

Algorithms for solving CSPs can be divided into two groups: search algorithms for solving CSPs and preprocessing algorithms for reducing futile searches; these are called *consistency algorithms.* Search algorithms for solving CSPs can be further divided into two groups: systematic tree search algorithms (backtracking) and iterative improvement algorithms (local search) [36].

Abundant literature reports a wide variety of techniques to solve CLPs, ranging from straightforward single-pass heuristics to local search [10] [15], genetic algorithms [11], tabu search [4], parallel methods [12] [20], linear programming and combinations of tree-search heuristics and dynamic programming [8] [25]. Two-dimension packing problems have also been surveyed [19]. In fact, perhaps an important advance is that literature has moved away from pure knapsacks to various additional restrictions, such as the gravity center [6] [8] [11], loading stability [5] [31], orientation restrictions [12] and loading bearing strength [1].

In particular, distributed Multi-agent Systems (MASs) are gaining more and more attention in *artificial intelligent* and *operation research* and have already found a home in many practical applications. An important prerequisite for their success is an ability to flexibly adapt their behavior via intelligent cooperation. Pawel Kalczynski used an agent-based approach to solve multi-container loading with grouping goals. But his approach did not consider three-dimensional models of shipping cargoes and containers; to keep things simple, only volume was taken into account [17].

1.2 Motivation of Theory of Constraints-based Management

The traditional algorithms mentioned above cannot be applied directly to solve complicated CSPs such as the *two-row pattern* of a CLP. Following is why.

1) Traditional algorithms usually seek an optimal solution to a problem. But restrictions involved in a really difficult problem are usually self-contradictory and hard to satisfy simultaneously; there may be no optimal solution to this problem.

2) There is never enough computation time for traditional algorithms to achieve an optimal solution on site.

A human expert, however, has experience solving conflicts among multiple objectives and restrictions involved in a CSP. The expert can usually define a problem in a more reasonable format by focusing on its constrained factors according to on-site situations. The problem can then be solved. Priorities of multiple objectives and restrictions can be classified so that a final reasonable solution can be sought within a reasonable computation time.

To simulate functions and flexibilities that come natural to a human expert, "Theory of Constraints-based Management" (TOCM) is proposed. With a human expert, divisions are made into hard and soft restrictions, according to the situations on site and user requests. Hard restrictions must be satisfied even though solution effectiveness is poor, and soft restrictions can be relaxed according to specific situations. This is done to prevent a system from running into a deadlock in which restrictions of self-contradiction cannot be satisfied simultaneously.

To make TOCM automatically find a final reasonable solution within a limited computation time, distributed MAS and the idea of Theory of Constraints (TOC) are introduced.

MAS is a system in which many intelligent agents interact with each other. The agents are considered autonomous entities. Their interactions can be either cooperative or selfish. That is, the agents can share a common goal, or they can pursue their own interests. In problems discussed in this research, in addition to competing for its own interest, the agent must also make concessions toward a common goal. The problems comprise when an agent should compete, when it should make concessions and to what extent. The answers are obtained from TOC.

TOC is an idea from management science invented by Dr. Eliyahu M. Goldratt [13]. The core idea in TOC is that every real system, such as a profit-making enterprise, must have at least one constraint, which is sometimes called a bottleneck. Because the constraint is a factor that limits the system from getting more of whatever it strives for, then a business manager who wants more profits must manage constraints. The techniques illustrated in TOC, such as Drum-Buffer-Rope (DBR) scheduling and the five-step focusing process for continuous improvement by focusing on constraints, can be applied to capacity constrained job shops with

high product diversity.

In TOCM, every agent is evaluated through an integrated evaluation function in order to reflect how much an agent satisfies its restrictions. It then competes while the value of its evaluation function is improved. The agent with the worst value of its integrated evaluation function is the constrained agent of a system. To achieve the common goal of a system, some agents need to make proper concessions for the common goal. Particular attention is focused on improving the constrained agent by striving for integrated improvement of its restrictions and making agents negotiate cooperatively. Ultimately, a final solution having the biggest profit is achieved.

As an example, this research attempts to solve the *two-row pattern* of a CLP. The experiment shows that TOCM based on a distributed and cooperative system is effective to solve CSPs in the real world by classifying restrictions and using TOC to build a kind of heuristic strategy.

1.3 Motivation of Three-zone Buffer Management

In TOCM, an agent is made better by improving the value of its integrated evaluation function; but one of its restrictions may be significantly violated even though its evaluation has been improved. Actually, a soft restriction is sometimes prevented from being relaxed infinitely. The weak point of TOCM is that it cannot guarantee that every soft restriction of an agent is well satisfied. To overcome this weak point, the "Three-zone Buffer Management" (TZBM) approach is introduced [32] and improved.

Just as with TOCM, TZBM classifies restrictions into hard and soft. But in TZBM, each soft restriction corresponds to a relaxable variable and the variable is given a relaxable range. The value of the relaxable variable can be used to reflect the extent to which its corresponding soft restriction is satisfied, and the given relaxable range is set by a human expert according to on-site situations and the expert's experience. Once values of relaxable variables are set within their own relaxable ranges, system agents can be divided into three zones (Green-zone, Yellow-zone and Red-zone) according to their own violating situations to soft restrictions. The constrained agent is in the Red-zone and one of its soft restrictions is violated farthest in the whole system. Through iterating agent negotiations and variable relaxations, a final satisfying solution is obtained until the Red-zone is emptied or relaxations reach maximum.

An ideal state in which all soft restrictions are violated as seldom as possible can be achieved through the mechanism with variable-relaxation. Further, the human expert can adjust the relaxable ranges and relaxation paces so that computation time can be controlled.

TZBM also attempts to solve the *two-row pattern* of a CLP. The experiment shows that TZBM works to eliminate violations to all soft restrictions of system

agents and seeks a *satisficing* instead of optimal solution with a limited computation time. It is comparable not only with TOCM, but also other optimization algorithms (Local search and Tabu search).

1.4 Motivation of Interaction-based Knowledge Acquiring Framework

The first motivation is to improve reusability and adaptability of an existing program. With the advancement of current computer technologies, it becomes possible to use the advantages of high-speed computation and the huge storage capacity of computers to assist human beings in solving difficult problems. Many researchers are resorting to varied optimization algorithms guided by heuristic control rules as feasible approaches. However, results obtained so far do not always satisfy user requirements, since the heuristic algorithm cannot always fully implement a human expert's flexibility and adaptability, which are revealed when the expert encounters a variety of dynamic or complex situations. That is to say, once a program is determined, the way to solve the current problem becomes fixed; i.e., strong adaptability is lost, which is the major characteristic in the way humans solve problems. Therefore, in developing practical applications, a program must have high reusability or flexibility and the user must be able to manipulate the program in a more flexible mode to improve efficiency.

The second motivation is for *knowledge succession* (i.e., knowledge and skills embodied in experienced workers are transferred to young workers). The year 2007 is the peak year when Japanese baby-boomers reach retirement age. The "Year-2007 problem" is upon us and the need for knowledge succession is now widely recognized. To many companies, the mass retirement of baby boomers will mean the loss of nearly uncountable highly skilled workers, still indispensable even in this age of automation and computerization. If legions of skilled workers retire, vast experience and cultivated know-how disappears.

Drawing out and using human expert knowledge and know-how must be investigated in depth, but it is a challenging job. When human experts encounter really difficult problems, they most often focus on critical aspects and neglect others. Even when changes occur in a really difficult problem, human experts can still deal with them flexibly. Experts select restrictions and prioritize objectives based on problem solving experience, but the know-how employed is difficult to capture. The expert, however, can demonstrate problem solving without difficulty. The most fundamental characteristic of a human expert is the interaction revealed when the expert encounters a novel problem. The problem may stimulate the expert to explore potential or latent skills, which are not always explicitly recognized by the expert. This type of knowledge is often called *tacit* knowledge. As Polanyi's [21]

definition, this type of knowledge is highly personal, context-specific, and hard to formalize and communicate. The human being starts to react when faced with particular problem solving circumstances.

In searching for a methodology to handle expertise using the computer, conventional expert system approaches were attempted from the 1970s to 1980s. These approaches failed because they concentrated on enlisting all knowledge apart from particular problem solving circumstances, and once the expertise was acquired and incorporated as the knowledge base in expert systems, that knowledge became fixed and could no longer change. On the other hand, the major property of human expertise is that it evolves and is dynamic. Therefore, once fixed, it may go unused.

The "Interaction-based Knowledge Acquiring Framework" (IKAF) is proposed in this research to develop an approach with strong adaptability by integrating a human expert agent with a computer agent. In IKAF, because much of a human expert's knowledge is experience-based and tacit, and it is quite difficult for us to implement that explicitly, the human expert agent is provided an opportunity to perform tacit skills in criticizing a result obtained using the conventional algorithm as well as the heuristics gained to that point within the computer agent. Then, from this performance, the computer agent detects the knowledge fragments that are commonly lacking in the existing knowledge-based system but are actually used in a human expert's performance. This is done by applying the deductive learning approach of explanation-based learning (EBL). Put simply, problem solving is done through cooperation between a computer agent and a human expert agent.

The procedures to transform a human expert's tacit knowledge into explicit knowledge are presented, and the examples of knowledge acquisition are demonstrated with respect to CLP general issues.

1.5 Future Research

As discussed in the sections above, the intelligence for solving problems is promoted step by step. But IKAF is still developing. Following are the minimum topics that remain to be discussed:

1) The transforming mode of tacit knowledge should be more intelligent in IKAF. As shown in this research, a control rule based on tacit knowledge and existing in the knowledge database can be activated at an appropriate moment. But a control rule based on tacit knowledge and not existing in the database should also be extracted automatically or semi-automatically by the EBL facility.

2) Computer algorithms embedded in the problem solver of IKAF should be extended. Experiments shown in this research are based on straightforward single-pass heuristic algorithms. And an attempt should be made to embed many solution-space-based algorithms such as Tabu Search, TZBM, to bring IKAF closer to practical use.

3) *Utility* problem. Learned control knowledge should not just be usable; it should also be useful. IKAF should extend the standard motion of operationality to include utility. In other words, IKAF not only requires that learned control rules be executable, but also requires that they actually improve the system's effectiveness and efficiency. For instance, the utility of a control rule can be defined as the cumulative improvement in search time attributable to the rule [22].

Chapter 2

Distributed and Cooperative System for solving Constraint Satisfaction Problem

2.1 Introduction

This chapter explains the methodology for solving *constraint satisfaction problem*s (CSPs) by approaches based on a distributed and cooperative system.

Section 2.2 briefly introduces a basic definition of the CSP, general algorithms and MAS-based algorithms developed to date for solving CSPs.

Section 2.3 introduces Theory of Constraints and its idea of Drum-Buffer-Rope as basic knowledge. Our "Theory of Constraints-based Management" (TOCM) is then proposed and discussed.

Based on TOCM, "Three-zone buffer management" (TZBM) is introduced in section 2.4. Its contribution is also emphasized.

In the final section 2.5, the illustrations for "Explanation-based Learning" and our "Interaction-based Knowledge Acquiring Framework" are discussed.

2.2 Former research

2.2.1 Definition of Constraint Satisfaction Problem

A CSP is a mathematical problem where one must find states or objects that satisfy a number of constraints (restrictions) or criteria. The CSP is the subject of intense research in both *artificial intelligence* and *operations research*.

As originally defined in artificial intelligence, constraints enumerate the possible values a set of variables may take. Informally, a finite domain is a finite set of arbitrary elements. A CSP on such domain contains a set of variables whose values

can only be taken from the domain, and a set of constraints, each constraint specifying the allowed values for a group of variables. A solution to this problem is an evaluation of the variables that satisfies all constraints. In other words, a solution is a way for assigning a value to each variable in such a way all constraints are satisfied by these values.

Formally, a CSP is defined a triple $\langle \mathbb{X}, \mathbb{D}, \mathbb{C} \rangle$, where $\mathbb{X}$ is a set of variables, $\mathbb{D}$ is a domain of values, and $\mathbb{C}$ is a set of constraints. Every constraint is in turn a pair $\langle t, \mathbb{R} \rangle$, where t is a tuple of variables and $\mathbb{R}$ is a set of tuples of values; all these tuples have the same number of elements; as a result $\mathbb{R}$ is a relation. An evaluation of the variables is a function from variables to $v : \mathbb{X} \to \mathbb{D}$. Such an evaluation satisfies a constraint $\langle (x_1, \ldots, x_n), \mathbb{R} \rangle$ if $(v(x_1), \ldots, v(x_n)) \in \mathbb{R}$. A solution is an evaluation that satisfies all constraints.

In reality, some constraints that are often used are expressed in compact form, rather than enumerating all possible values of the variables. The constraint expressing that the values of some variables are all different is one of the most used such constraints.

Problems that can be expressed as CSPs are the Eight queens puzzle, the Sudoku solving problem, the Boolean satisfiability problem, scheduling problems and various problems on graphs such as the graph coloring problem [33].

Given a CSP instance, the goal might usually be as follows:

1) Determine whether the instance has any solutions.

2) Find any solution.

3) Find all solutions.

4) Find solution that maximizes/minimizes some given objective function: *combinatorial optimization*.

2.2.2 Solving Constraint Satisfaction Problem

Algorithms for solving CSPs can be divided into two groups: search algorithms for finding a solution of CSPs and preprocessing algorithms for reducing futile search, which are called *consistency algorithms*. Search algorithms for solving CSPs can be further divided into two groups: systematic tree search algorithms (backtracking) and iterative improvement algorithms (local search).

Backtracking is a recursive algorithm. It maintains a partial assignment of the variables. Initially, all variables are unassigned. At each step, a variable is chosen, and all possible values are assigned to it in turn. For each value, the consistency of the partial assignment with the constraints is checked; in case of consistency, a recursive call is performed. When all values have been tried, the algorithm backtracks. In this basic backtracking algorithm, consistency is defined as the satisfaction of all constraints whose variables are all assigned. Several variants of backtracking exist. Backmarking improves the efficiency of checking consistency. Backjumping allows saving part of the search by backtracking "more than one variable" in some

cases. Constraint learning infers and saves new constraints that can be later used to avoid part of the search. Look-ahead is also often used in backtracking to attempt to foresee the effects of choosing a variable or a value, thus sometimes determining in advance when a subproblem is satisfiable or unsatisfiable.

Local search methods are incomplete satisfiability algorithms. They may find a solution to a problem, but they may fail even if the problem is satisfiable. They work by iteratively improving a complete assignment over the variables. At each step, values of a small number of variables are changed, with the overall aim of increasing the number of constraints satisfied by this assignment. In reality, local search appears to work well when these changes are also affected by random choices. Integration of search with local search has been developed, leading to hybrid algorithms.

Consistency techniques were first introduced by researchers in artificial intelligence [34] to improve the efficiency of picture recognition programs, by . Picture recognition involves labelling all the lines in a picture in a consistent way. The number of possible combinations can be huge, while only very few are consistent. Consistency techniques effectively rule out many inconsistent labellings at a very early stage, and thus cut short the search for consistent labellings. These techniques have since proved to be effective on a wide variety of hard search problems. Notice that consistency techniques are deterministic, as opposed to the search, which is non-deterministic. Thus the deterministic computation is performed as soon as possible and non-deterministic computation during search is used only when there is no more propagation to be done. Nevertheless, the consistency techniques are rarely used alone to solve CSPs completely (but they could be).

2.2.3 Solving Constraint Satisfaction Problem by Distributed Multi-agent System

The basic concepts of agent and distributed Multi-agent System (MAS) are reviewed simply before their application to CSPs.

The notion of agent is found in a wide range of research in Computer Science and Distributed Artificial Intelligence. According to Russell's definition, an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators. A software agent receives keystrokes, file contents and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets [26].

MAS comprises of multiple, interacting agents. As a distributed system, the distributed MAS method for solving problems has the following advantages [14]:

1) can simplify problem solving by splitting the problem into simple tasks.

2) can tolerate uncertain data and knowledge.

3) offers conceptual clarity and simplicity of design.

4) presents graceful degradation in computational complexity.

5) allows incremental modification of the system boundary.

6) is well suited to distributed problems.

A distributed CSP is always solved by the methods of distributed MAS. A distributed CSP is a CSP in which the variables and constraints are distributed among automated agents. Finding a value assignment to variables that satisfies inter-agent constraints can be viewed as achieving coherence or consistency among agents. Achieving coherence or consistency is one of the main research topics in MAS. Therefore, distributed constraint satisfaction techniques can be considered an important infrastructure for cooperation. Various application problems in MAS can be formalized as distributed CSPs, by extracting the essential part of the problems such as recognition problems, resource allocation problems, multi-agent truth maintenance tasks, and scheduling /timetabling tasks.

Some search algorithms for solving distributed CSPs are *asynchronous backtracking* algorithm, *asynchronous weak-commitment search* algorithm and *distributed breakout* algorithm. In the *asynchronous backtracking* algorithm, agents act concurrently and asynchronously without any global control, while the completeness of the algorithm is guaranteed. In the *asynchronous weak-commitment search* algorithm, when an agent cannot find a value consistent with highest priority agents, the priority order is changed so that the agent has the highest priority. As a result, when an agent makes a mistake in selecting a value, the priority of another agent becomes higher; accordingly the agent that made the mistake will not commit to the bad decision, and the selected value is changed. In the *distributed breakout* algorithm, neighboring agents exchange values that provide possible improvements so that only an agent that can maximally improve the evaluation value can change its variable value, and agents detect quasi-local-minima instead of real-minima.

Distributed-ATMS (assumption-based truth maintenance system) comprises a kind of distributed consistency algorithms. In this distributed problem-solving model, each agent has its own ATMS, and these agents communicate hypothetical inference results and nogoods among themselves [36].

In addition, some traditional CSPs that have been attempted to be solved by non-MAS methods are also being investigated by the MAS-based methods because of inherent MAS strengths. For example, David He solved a manufacture scheduling problem using MAS-based methods [14]. And Pawel Kalczynski solved multi-container loading with grouping goals using an agent-based approach [17].

Figure 2.1: Illustration for Drum-Buffer-Rope of Theory of Constraints

2.3 Theory of Constraints-based Management

2.3.1 Theory of Constraints

Before discussing our proposed approaches, the methodology for Theory of Constraints (TOC) is introduced here.

TOC is an idea from management science invented by Dr. Eliyahu M. Goldratt. The core idea in TOC is that every real system such as a profit-making enterprise must have at least one constraint, which is sometimes called a bottleneck. Because the constraint is a factor that limits the system from getting more of whatever is strives for, then a business manager who wants more profits must manage constraints [13].

The techniques illustrated in TOC such as the Drum-Buffer-Rope scheduling (DBR) and the five-step focusing process for continuous improvement by focusing on the constraints can be applied to capacity constrained job shops with high product diversity.

In the DBR scheduling, *"Drum"* refers to the pace followed, *"Buffer"* refers to the safeties used to exploit constraints, and *"Rope"* refers to interdependent events that trigger each other's activities. In details, DBR involves flying a rope between the leading scout and the slowest scout in a line of hikers (i.e., the constraint) as shown in Figure 2.1. This solution constrains the scouts in front of the slowest scout to walk no faster on average than the slowest scout. That is to say, even the leading scout can walk faster, but for the profit of the whole troop, he must limit his speed to subordinate the slowest scout, and this behavior is called concession. More important, the rope prevents the work-in-process inventory in front of the slowest scout from growing beyond the slack allowed by the length of the rope. By leaving slack in the rope, a protective buffer will open up just in front of the slowest scout.

A drum is a strategic operation that has limited resources and determines the flow of work through the system. A system can go only as fast as the slowest or the most overloaded resource. This is called constrained resource, which sets the pace of the system. A buffer is a pocket of time represented by work in process that is reserved ahead of the drum, the constrained resource. The rope is the length of time needed to accomplish processes ahead of the drum.

For different practical problems, DBR may implicate different objects, but the central idea of DBR does not change. How to determine the length of the rope is always very important and difficult. The length of the rope is the size of the work-in-process and restricts that from growing indefinitely. Sizing the buffer is difficult because of the following dilemma. Undersizing the buffer will leave the constrained capacity resource open to starvation and lost throughput for the entire plant, while oversizing the buffer will increase operating expenses and cycle time and decrease inventory turns, resulting in decreased cash flow. The right size of the buffer is usually determined by monitoring the buffer in practice.

On the other hand, the five-step focusing process provides the foundation for many of TOC's generic solutions, which include the management of processes, inventory, supply chains, product development and projects (single and multiple), personnel and decision-making. It is described as follows [29]:

Step1, identify the system's constraint(s).

Step2, decide how to exploit the system's constraint(s).

Step3, subordinate/synchronize everything else to the above decisions.

Step4, elevate the system's constraint(s).

Step5, if in the above steps the constraint has shifted, go back to Step1, and do not allow inertia to become the system's constraint.

2.3.2 Approach Description

As discussed above, the original idea of TOC is an overall management philosophy that aims to continually achieve more of a system's goals. It has been widely used in all aspects of logistics: operations, finance and measurements, project management and engineering, distribution and supply-chain. But it has seldom been used in the optimization algorithm. Our proposed approach in this research is called "Theory of Constraints-based Management" (TOCM) because it is an extension of TOC.

In TOCM, restrictions are divided into hard and soft to decrease deadlocks and increase flexibility of the final solutions. Hard restrictions must be satisfied even though solution effectiveness is poor; soft restrictions may be relaxed according to specific situations.

An original problem is formatted as distributed MAS. Each agent is evaluated through an integrated evaluation function. The integrated evaluation function consists of several sub-evaluation functions and each sub-evaluation function corresponds to a concrete soft restriction to be considered. The bigger the violation, the bigger the value. Further, each sub-evaluation function is set up with a weighted parameter that shows the importance of the soft restriction. The agent with the maximum evaluation function is the worst agent.

There are three kinds of cooperative negotiation operations among agents. "Self-adjusting" is to adjust positions of one agent's resources. "Swapping" is to swap

resources of two agents. "Inserting" is to remove a resource from one agent and insert it into another agent.

The solving process of TOCM comprises two stages. The first stage is to generate an initial solution with only the hard restrictions taken into account, and the second stage is to improve the initial solution with soft restrictions satisfied to the greatest extent possible. Specifically, the cooperative negotiation strategy of the improving stage can be described as follows: the worst agent is the constraint whose capacity is elevated by direct and/or indirect agent negotiations; negotiations between unconstrained agents are also conducted to eliminate the constraint while unconstrained agents can be improved simultaneously. This strategy, which is also a kind of heuristic local search, guarantees that a system bottleneck is always solved preferentially, which should make negotiations more efficient.

2.4 Three-zone Buffer Management

2.4.1 Approach Description

In TZBM, restrictions are classified as hard and soft, just as with TOCM. But each soft restriction corresponds to a relaxable variable and the value of the relaxable variable can be used to reflect the extent to which its corresponding soft restriction is satisfied.

All agents are divided into three zones: Green-zone, Yellow-zone and Red-zone according to how well the soft restrictions are satisfied. The agents satisfying the original restrictions of all soft restrictions are assigned to the Green-zone; the agents satisfying the relaxed restrictions of all soft restrictions are assigned to the Yellow-zone; and the agents failing to satisfy even the relaxed restrictions of any restriction are assigned to the Red-zone. Each agent is still evaluated through an integrated evaluation function. More important, TZBM makes the evaluation method consistent with the zone dividing criterion. If an agent belongs to the Red-zone and has the maximum (worst) value of sub-evaluation corresponding to unsatisfied soft restrictions, then it is called the constrained agent.

In TZBM, agents can do negotiation operations such as "Self-adjusting," "Swapping" and "Inserting."

TZBM still includes two stages in its solving process. The cooperative negotiation strategy of the improving stage can be described as follows: in the initial state, relaxable variables are set to their smallest values and then strict criteria are set to judge which agents are in the Red-zone. Agents in the Red-zone are then improved by striving for a trade-off of soft restrictions and making agents negotiate cooperatively. If agents in the Red-zone cannot be eliminated, soft restrictions are relaxed by enlarging their corresponding relaxable variables and the three zones are redivided. In so doing, constrained agents can be eliminated under a looser envi-

ronment. By iterating the above process, a final satisficing solution is obtained and an ideal situation is sought in which all soft restrictions are violated as seldom as possible.

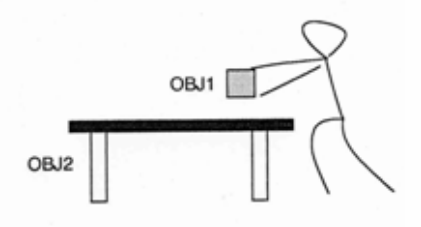
2.4.2 Contribution of Three-zone Buffer Management

Our approaches are constructed with the goal of satisficing[1] rather than optimizing [28]. Simon was the first prominent theorist to doubt that people are able to calculate the optimal choice. He believed that it is impossible for people to consider all options and all the information about those items that the conventional normative decision models assume. Simon proposed his own model of decision making as an alternative to the optimizing approach. He called his proposal the "satisficing" decision model. It implies that people think of options, one by one, and choose the first course of action that meets or surpasses some minimum criterion that will satisfy them. Simon believed that decision makers establish a criterion (their "level of aspiration") that an alternative must meet in order to be acceptable. People examine possible options in the order that they think of them. Eventually, they accept the first option that meets their criterion. Simon pointed out that human beings lack the cognitive resources to maximize: we usually do not know the relevant probabilities of outcomes, we can rarely evaluate all outcomes with sufficient precision, and our memories are weak and unreliable.

A more realistic approach to rationality takes into account these limitations, which is called *bounded rationality*. In interviewing the experts on site and observing their ways of allocating cargoes into containers, we recognized that they are typically based upon Simon's sacrificing principle. Just as with Simon's idea of level of aspiration, experts on site are apt to reconsider the rigorousness of the restrictions of the CSP. Beyond that, they have the flexibility to relax those restrictions so that they can find satisficing solutions. Their decisionmaking is quite different from optimizing and deliberating, and depends greatly on what options are available at hand and on whether or not they meet their relaxed restrictions. This is the rationale of our proposed algorithm, and we implement this idea using MAS and DBR.

[1]The word satisficing was coined by Herbert Simon as a portmanteau of "satisfy" and "suffice."

Target concept definition:
SAFE-TO-STACK(x,y) iff NOT (FRAGILE(y)) or LIGHTER(x,y)
Training example:
ON(OBJ1, OBJ2)
ISA(OBJ1, BOX)
ISA(OBJ2, ENDTABLE)
COLOR(OBJ1, RED)
COLOR(OBJ2, BLUE)
VOLUME(OBJ1, 1)
DENSITY(OBJ1, 0.1)



Domain theory:
VOLUME(p1,v1) and DENSITY(p1,d1) WEIGHT(p1,v1*d1)
WEIGHT(p1,w1) and WEIGHT(p2,w2) and LESS(w1,w2) LIGHTER(p1,p2)
ISA(p1,ENDTABLE) WEIGHT(p1,5) [default]
LESS(1,5)
Operationality criterion:
The learned concept must be expressed in terms of the predic ates used to describe examples.

Learned concept:
VOLUME(x,v1) and DENSITY(x,d1) and LESS(v1*d1, 5) and ISA(y, ENDTABLE)
SAFE-TO-STACK(x,y)

Figure 2.2: An example of Explanation-based Learning

2.5 Interaction-based Knowledge Acquiring Framework

2.5.1 Explanation-based Learning

Before discussing our proposed approach, the methodology of EBL is introduced in this section.

Explanation-based learning (EBL) is a technique by which an intelligent system can learn by observing examples. EBL systems are characterized by the ability to create justified generalizations from single training instances. They are also distinguished by their reliance on background knowledge of the domain under study. Although EBL is usually viewed as a method for performing generalization, it can be viewed in other ways as well. In particular, EBL can be seen as a method that performs four different learning tasks: generalization, chunking, operationalization, and analogy [9].

Let us explain the functions of the EBL using an original example illustrated in Mitchell's book "Machine Learning" [23] so as to make it easy to comprehend "EBL Facility" well. Mitchell et al. consider the target concept SAFE-TO-STACK(x, y), that is, the set of object pairs $< x, y >$ such that x can be safely stacked on y. The target concept definition, training instance, theory, and operationality cri-

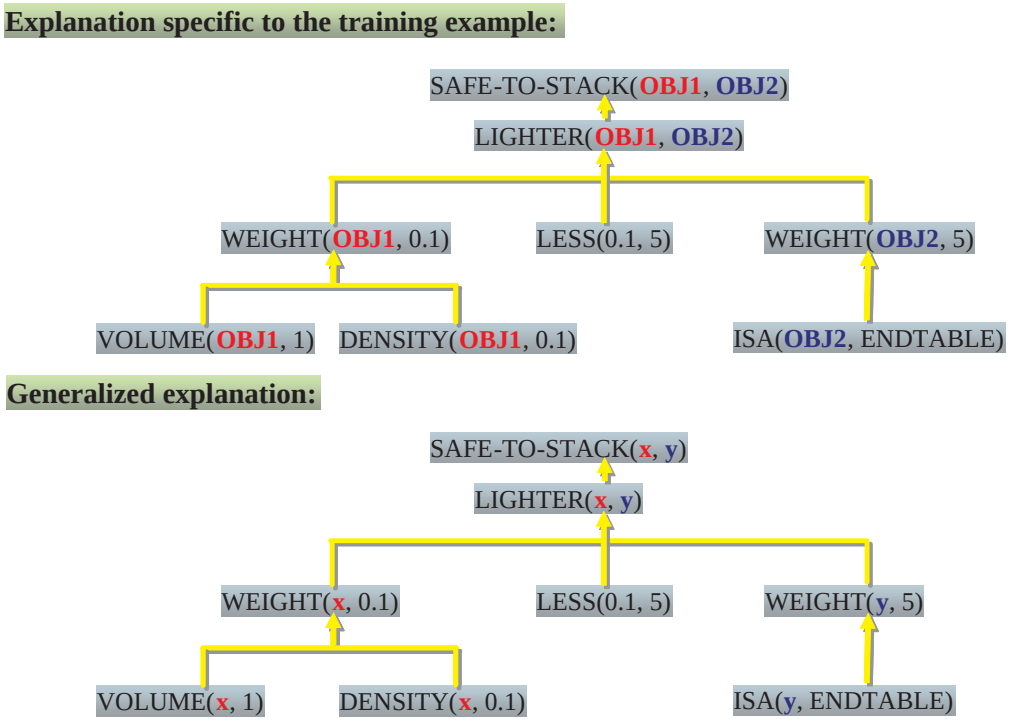


Figure 2.3: Explanation Tree in the example of Explanation-based Learning

terion are given in Figure 2.2.

The definition of SAFE-TO-STACK specifies that an object can be safely stacked on a second object if the second object is not fragile or the first object is lighter than the second. The domain theory encapsulates the system's knowledge about objects, weight, etc. The training example illustrates an instance of two objects, OBJ1 and OBJ2, that can be safely stacked on top of each other. Finally, the system's operationality criterion specifies that the explanation must be expressed in terms of easily evaluated predicates.

The following covers how explanation-based generalization (EBG) learns from this example. EBG proves that OBJ1 is SAFE-TO-STACK on OBJ2 (see Figure 2.3). The proof is then generalized by regressing the target concept through the proof structure. The regression process replaces constants with variables while preserving the structure of the proof. The purpose of regression is to find the weakest conditions under which the proof structure will hold. In this manner EBG produces the following sufficient conditions for describing the concept SAFE-TO-STACK: $VOLUME(x, v_1)$ and $DENSITY(x, d_1)$ and $LESS(v_1 \times d_1, 5)$ and $ISA(y, ENDTABLE)$. These conditions specify that x can be safely stacked on y if y is an endtable and the volume times the density of x is less than 5. Notice that the domain theory specifies that all endtables weigh 5 lbs by default. This description satisfies the operationality

criterion and is justified by the proof.

2.5.2 Approach Description

Here we would like to insist that the purpose of a knowledge system should shift from encoding knowledge to preserving human expertise as well as distributing knowledge from experts to others. Preferably, we think opportunities for experts to apply their knowledge should be offered by the system and the methodology should assist the knowledge elicitation process by stimulating the expert's proactive and voluntary problem solving efforts. In this way, it is most important that a platform should help an expert derive and transform tacit knowledge into explicit knowledge. In other words, this is a shift from aiming at automated knowledge extraction to designing a work environment where knowledge is not only visualized but used by experts and others. Appropriate assistance is provided by the system using its evolvable knowledge.

Another important issue for designing such a computer platform is how to integrate human expertise produced through practice. It must encompass the abilities of machine intelligence of optimization techniques that can handle and formulate various heuristic algorithms. A possible solution is to abstract the expert's practice into some form of knowledge and then incorporate it into the heuristic algorithm to improve flexibility of the general algorithm.

Based on the above considerations, in this research, we propose a framework called Interaction-based Knowledge Acquiring Framework (IKAF) and we introduce EBL as a core learning function embedded within the system. EBL interacts with a human expert and acquires the expert's tacit knowledge.

EBL is a deductive machine learning method that uses *domain theory* as domain specific knowledge. EBL can generalize a specific explanation formed for a particular training example and derive generalized knowledge from that by analyzing why that example is an instance of a concept to be learned. The methodology of EBL yields a feasible and flexible way to abstract the expert's knowledge. In our framework, the expert is allowed to revise the result calculated by the conventional optimization techniques and to take this revision as an example for EBL. Through the analysis of this training example, the expert's tacit *control knowledge* is learned and can be used in the next calculation based on the revised knowledge base, to which the acquired fragment of an expert's tacit knowledge is added incrementally.

The computer agent and the human expert agent cooperate to solve difficult problems. The computer agent undertakes the heavy computation tasks. The human agent undertakes the softening and adjusting tasks.

Chapter 3

Theory of Constraints-based Management

3.1 Introduction

"Theory of Constraints-based Management" (TOCM) can be used to solve many difficult problems such as the container loading problem (CLP), which comes from the centers of physical distributions, manufactories and warehouses. The *two-row pattern* of a CLP mentioned in the introduction chapter (Chapter 1) will be solved in this chapter for demonstrating how TOCM solves a complicated *constraint satisfaction problem* (CSP).

In section 3.2, the definition of the *two-row pattern* and its evaluation functions are introduced.

Then the loading process of the *two-row pattern* is discussed in detail in the section 3.3.

The computation experiment is shown in the section 3.4, and finally the conclusion in the section 3.5.

3.2 Definition of Container Loading Problem

In the *two-row pattern* of a CLP, all cargoes are rectangular parallelepipeds of known sizes and weights. Each cargo is positioned parallel to the side walls of a container. We assume origins of containers and cargoes are their innermost points respectively from the perspective shown in Figure 1.1. The length/width/height of a container is with respect to x-axis/y-axis/z-axis, respectively.

The *two-row pattern* of a CLP is a multiple destination delivering problem. There are N_o ($N_o \geq 1$) packing orders to be allocated into modules then containers, and then those are delivered to the destinations on a trip. Consequently all

cargoes are grouped into N_o groups. These packing orders should obey the rule of "first-in-last-out" (FILO) according to the delivering destinations and routes.

A container is defined as $C = \{L, W, H, CC\}$, where these included parameters are the length, width, height and carrying capacity of a container, respectively. In this research, only one container type is used for each data and the container number is not limited.

$N_m (N_m \geq 1)$ indicates the module number in a loaded container and it is decided by the algorithm.

Let $m_k = \{ml_k, mw_k, mh_k, mg_k, mx_k, my_k\}$ represents the k-th module in a container (ml_k: length, mw_k: width, mh_k: height, mg_k: weight, mx_k/my_k: the position along x-axis/y-axis in the container). The ml_k/mw_k is the maximum length/width of all included cargoes of m_k. The mh_k and mg_k are calculated by summing the height and weight of each included cargo respectively. The mx_k and my_k are set and adjusted in the algorithm. And by default the side ml_k of m_k is allocated along the W direction of the container.

Hard Restrictions:

- $\mathbf{H}_1$: The packing orders must obey FILO.
- $\mathbf{H}_2$: The total weight of modules loaded in a container cannot exceed the carrying capacity of the container, i.e.

$$\sum_{k=1}^{N_m} mg_k \leq CC \tag{3.1}$$

- $\mathbf{H}_3$: Each module lies completely within a container.

Soft Restrictions:

- $\mathbf{S}_1$: From the viewpoint of transporting and handling a loaded container - such as lifting it onto a ship - it is desirable that the gravity centre of a loaded container is close to the geometrical mid-point of its floor (if the weight is distributed very unevenly, certain handing operations may be impossible to carry out).
  When the gravity center of a container floor (X_{gc}, Y_{gc}) is defined as

$$\begin{cases} X_{gc} = \frac{\sum_{k=1}^{Nm}[(mx_k + mw_k/2) \times mg_k]}{\sum_{k=1}^{Nm} mg_k} \\ Y_{gc} = \frac{\sum_{k=1}^{Nm}[(my_k + ml_k/2) \times mg_k]}{\sum_{k=1}^{Nm} mg_k} \end{cases} \tag{3.2}$$

  , the sub-evaluation for the gravity center E_{S_1} can be evaluated as:

$$E_{S_1}(C) = E^x_{S_1}(C) + E^y_{S_1}(C) \tag{3.3}$$

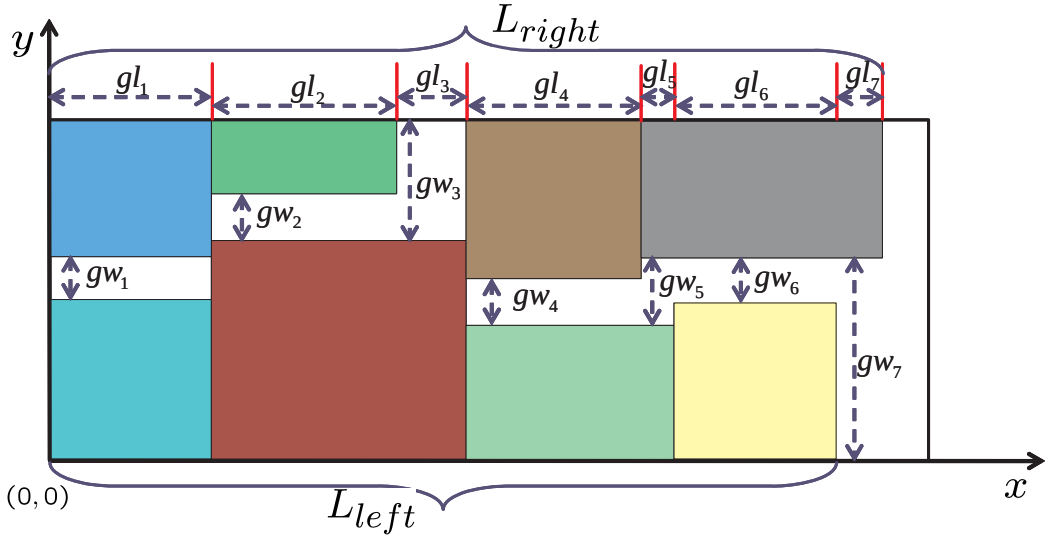


Figure 3.1: Inside seven kinds of gaps (topview)

where

$$E^x_{S_1}(C) = \begin{cases} 0, & |X_{gc} - L/2| \leq (\alpha \times L) \\ \frac{|X_{gc}-L/2|}{L}, & otherwise \end{cases} \tag{3.4}$$

$$E^y_{S_1}(C) = \begin{cases} 0, & |Y_{gc} - W/2| \leq (\beta \times W) \\ \frac{|Y_{gc}-W/2|}{W}, & otherwise \end{cases} \tag{3.5}$$

(α and β are defined according to the user's command. In this research, $\alpha \leftarrow 0.05$ and $\beta \leftarrow 0.05$.)

- $\mathbf{S}_2$: For making good use of container space, the floor occupied ratio E_{S_2} is evaluated as:

$$E_{S_2}(C) = 1 - \frac{\sum_{k=1}^{N_m}(ml_k \times mw_k)}{L \times W} \tag{3.6}$$

- $\mathbf{S}_3$: The good balance between the left and right rows is for fixing the modules on the tails of two rows. The sub-evaluation for the balance is defined as

$$E_{S_3}(C) = \frac{|L_{left} - L_{right}|}{mw_{min}} \tag{3.7}$$

where L_{left}/L_{right} is the total length of the left/right row as shown in Figure 3.1 and mw_{min} is the minimum width of all modules.

- $\mathbf{S}_4$: Module crossing packing (MCP) is shown in Figure 3.1 (there are seven gaps in this sample), which means the wide module is corresponding to the narrow one and the gap is left in the middle. It is desired that the gap widths

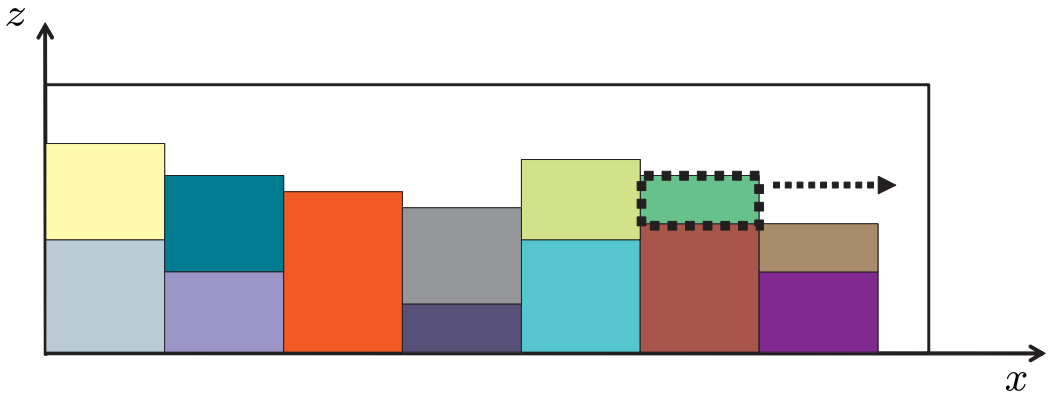


Figure 3.2: Unstable module (left-hand sideview)

of are close to regular widths because of the easiness of filling cushions into gaps. The sub-evaluation for MCP is defined as:

$$E_{S_4}(C) = \frac{\sum_{k=1}^{N_g} \frac{|gw_k - gw_{reg}|}{W - ml_{min}} \times gl_k}{\sum_{k=1}^{N_g} gl_k} \tag{3.8}$$

where gw_k is the width of the k-th gap between two modules, gl_k is the length of the k-th gap, N_g is the total number of gaps, gw_{reg} is a regular distance between two rows required by the user (in this research, $gw_{reg} \leftarrow 30cm$) , and ml_{min} is the minimum length of all modules.

- S_5: In order to keep the stability of the module, the cargo at the top of the module should be restricted by adjacent cargoes against cargo movements. For example the cargo surrounded by the dot line in Figure 3.2 is an unstable cargo because it is easy sliding backward (positive direction of x-axis as shown in Figure 3.2) and falling down from the left row. The stability evaluation of two adjacent modules m_{k-1} and m_k is expressed as

$$E'_{S_5}(m_{k-1}, m_k) = \begin{cases} \frac{mh_{k-1} - h_{k-1}^{top}/2 - mh_k}{mh_{av}}, & (mh_{k-1} - h_{k-1}^{top}/2) > mh_k \\ \frac{mh_k - h_k^{top}/2 - mh_{k-1}}{mh_{av}}, & (mh_k - h_k^{top}/2) > mh_{k-1} \\ 0, & otherwise \end{cases} \tag{3.9}$$

where mh_{av} is the average height of all cargoes, h_{k-1}^{top}/h_k^{top} is the height of the top cargo of m_{k-1}/m_k. The sub-evaluation for the stability of the modules is:

$$E_{S_5}(C) = \frac{\sum_{rw=left}^{right} \sum_{k=2}^{N_m^{rm}} E'_{S_5}(m_{k-1}, m_k)}{N_m^{left} + N_m^{right} - 2} \tag{3.10}$$

where N_m^{left}/N_m^{right} is the module number of the left/right row ($N_m^{left} + N_m^{right} > 2$).

Based on these sub-evaluation functions, the integrated evaluation function for a container can be described as:

$$E(C) = \sum_{i=1}^{5} u_i \times E_{S_i}(C) \tag{3.11}$$

where $u_i (i = 1, 2, ..., 5)$ are the weighted parameters, which depend on how much the user prioritizes in relation with others [1].

3.3 Solving by TOCM

The loading process is divided to two parts, the first part is to generate the initial solution and the second part is to improve the initial solution.

3.3.1 Generating initial solution

The basic loading procedure is considered as a 2-dimension allocating problem and it is difficult to obtain a minimum container number because of properties of modules and complicated restrictions. To get the minimum container number is not the purpose in this research, but the main purpose is to improve the constrained agents and to avoid the solving process from falling down a deadlock situation. So a concision but practical algorithm is utilized here.

The initial solution of the *two-row pattern* of a CLP is generated as follows (Note: it is assumed that the length of a module is longer than its width initially and its ml_k side is allocated along the y-axis of a container) :

- **Step1**: (Arrange packing orders) Arrange all given modules according to their packing orders and heights in a series $\mathbb{M} = \{ m_1^1 , m_2^1 , ... , m_{n_1}^1, ... , m_1^j , ... , m_i^j , ... , m_{n_j}^j, ... , m_1^{N_o} , ... , m_{n_{N_o}}^{N_o} \}$, where N_o is the total number of packing orders (there are n_j modules in the j-th order group). Let $r \leftarrow 1$.

- **Step2**: (Choose a container) Choose an empty one $C_r(L_r, W_r, H_r, CC_r)$ and let $L_{left} \leftarrow 0$, $L_{right} \leftarrow 0$.

[1] If the floor occupied ratio of the last container is low (e.g., less than 50%), all of its weighted parameters are set to be zero and it proposes no negotiation. Otherwise its u_1 and u_2 (corresponding to the gravity center and the floor occupied ratio) are set to be zero for that the last container is not always identified as the system constraint.

- **Step3**: (Choose a module in the case of $L_{left} = L_{right}$) If there is no module satisfying *packing condition(1) and (2)* in the front order group, go to *Step5*, else choose the first module m_i^j, place it into the left row and let $L_{left} \leftarrow L_{left} + mw_i$, $C_r \leftarrow C_r + \{m_i^j\}$, $\mathbb{M} \leftarrow \mathbb{M} - \{m_i^j\}$).

- **Step4**: (Choose a module in the case of $L_{left} \neq L_{right}$) Let L_{short}/L_{long} be the shorter/longer of L_{left} and L_{right}.
  If there is a set of modules satisfying *packing condition(1)-(3)* in the front order group (considering the rotation states of modules), choose a module m_i^j from the set with a good use of the space, place it in the shorter row, let $L_{short} \leftarrow L_{short} + mw_i$, $C_r \leftarrow C_r + \{m_i^j\}$, $\mathbb{M} \leftarrow \mathbb{M} - \{m_i^j\}$) and go back to *Step3*. Else if there is a set of module satisfying *packing condition(1) and (4)* in the front order group, choose the first module m_i^j, place it in the shorter row after a small gap (e.g., the gap [gl_3,gw_3] in Figure3.1), let $L_{short} \leftarrow L_{long} + mw_i$, $C_r \leftarrow C_r + \{m_i^j\}$, $\mathbb{M} \leftarrow \mathbb{M} - \{m_i^j\}$ and go back to *Step4*. Else no module can be loaded in the current container, go to *Step5*.

- **Step5**: (Judge the end condition) If $\mathbb{M} = \Phi$, the initial solution is generated and end the generating procedure, else let $\mathbb{C} \leftarrow \mathbb{C} + \{C_r\}$, $r \leftarrow r + 1$ and go back to *Step2*.

packing condition: There is a module m_i^j so that:
1) its weight satisfies $mg_i \leq CC_r - \sum_{k=1}^{N_m} mg_k$
2) its width satisfies $mw_i \leq L_r - min\{L_{left}, L_{right}\}$
3) its length satisfies $ml_i \leq W_r - ml_i^{'}$, where $ml_i^{'}$ is the most length of modules located at the opposite row
4) its width satisfies $mw_i \leq L_r - max\{L_{left}, L_{right}\}$

According to the above generating procedure, the initial solution is obtained and described as $\mathbb{C} = \{C_1, C_2, ..., C_{N_c}\}$.

3.3.2 Adjusting heights of modules

Sometimes, because weights of the cargoes are too heavy, the occupied ratios of most of loaded containers are quite low, which makes the container stability be worse. In this case, parts of modules need to be adjusted.

Generally, it is to decrease the heights of the modules by calling on the first process of making modules [32] and how much high to be decreased depends on the requirement of the floor occupied ratio. The criterion of the floor occupied ratio is given by the expert on site.

3.3.3 Improving initial solution

First, some key variables of the negotiation procedure are listed as follows:

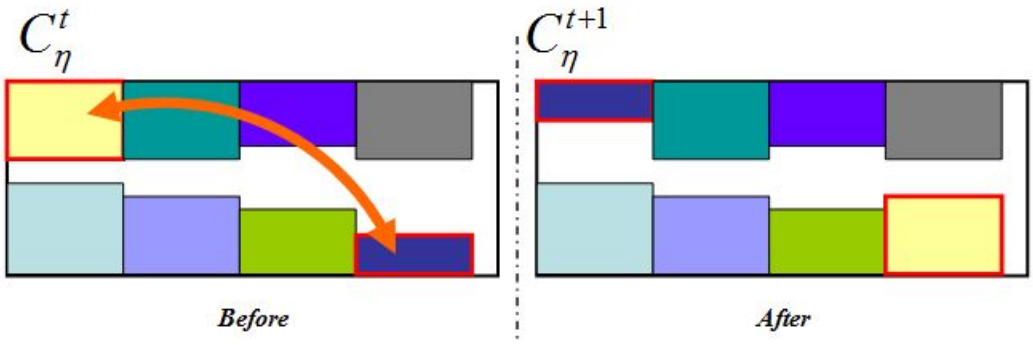


Figure 3.3: Improving solution by self-adjusting operation (topview)

- t: the current iteration number;
- t^*: the maximum iteration number (depending on the given computation time);
- C_γ: the constrained container;
- C_η^t: the container proposing the negotiation conditions;
- C_δ^t: the cooperative container, which responds to the negotiation conditions proposed by C_η^t;
- $\mathbb{C}_\delta$: all containers that are possible to be C_δ^t for current C_η^t;
- $\mathbb{T}_\mathbb{C}$: the tabu list whose element was selected as C_η^t ever but could not be adjusted;
- $\mathbb{T}_\mathbb{M}$: all modules having been attempted in C_η^t;

After C_γ^t being identified, three kinds of negotiation operations can be attempted in order to improve C_γ^t.

After C_η^t being identified, three kinds of negotiation operations can be attempted.

"Self-adjusting": to change the positions of its resources (modules) as shown in Figure 3.3.

"Swapping": to exchange its resources with other cooperative containers as shown in Figure 3.4.

"Inserting": to insert resources from other cooperative containers as shown in Figure 3.5.

All negotiation operations are carried out in the same order group.

At the initial status, let $t \leftarrow 0$, $\mathbb{C}_\delta \leftarrow \Phi$, $\mathbb{T}_\mathbb{C} \leftarrow \Phi$ and $\mathbb{T}_\mathbb{M} \leftarrow \Phi$. If all modules of C_η^t are put into $\mathbb{T}_\mathbb{M}$ (i.e. $C_\eta^t = \mathbb{T}_\mathbb{M}$) after even all negotiation operations having been attempted, C_η^t should be added into $\mathbb{T}_\mathbb{C}$. If $C_\eta^t = C_\gamma$ and C_η^t is improved, $\mathbb{T}_\mathbb{C}$ needs to be cleaned up at once. If there is a successful negotiation between two

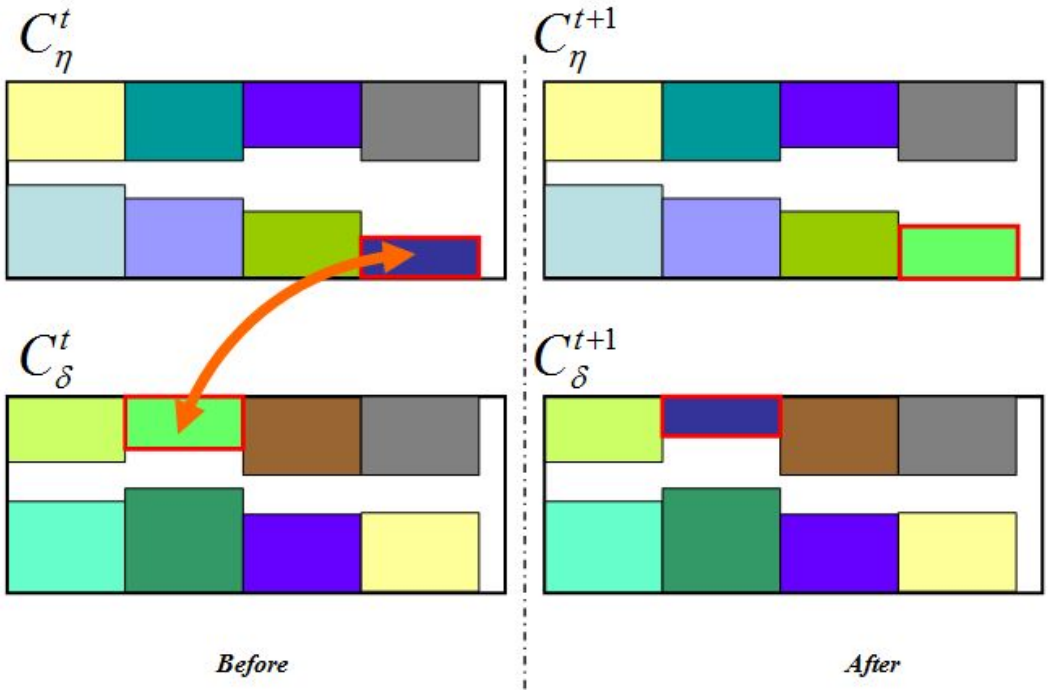


Figure 3.4: Improving solution by swapping operation (topview)

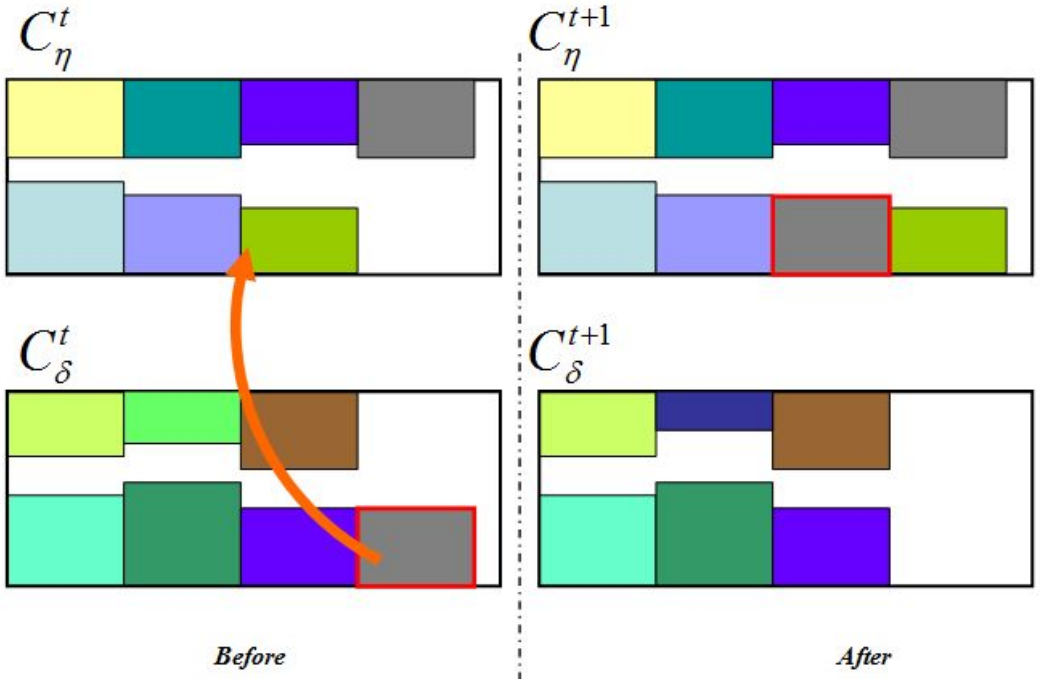


Figure 3.5: Improving solution by inserting operation (topview)

unconstrained containers, C_γ should be released from $\mathbb{T}_\mathbb{C}$ in order to attempt to elevate the constraint once again.

Second, core steps of the negotiation are described as follows:

- *Step1*: (Identify the system's constraint) Choose a container C_γ with the maximum value of the evaluation function from $\mathbb{C}$ as the current system's constraint.

- *Step2*: (Decide the negotiation proposer) In the case of $\mathbb{C} = \mathbb{T}_\mathbb{C}$, go to *Step8*. In the case of $\mathbb{C} \neq \mathbb{T}_\mathbb{C}$, (a) if $C_\gamma \notin \mathbb{T}_\mathbb{C}$, let $C_\eta^t \leftarrow C_\gamma$, else choose a container randomly from $\mathbb{C} - \mathbb{T}_\mathbb{C}$ as C_η^t, and (b) if $\mathbb{C}_\delta = \Phi$, let $\mathbb{C}_\delta \leftarrow \mathbb{C} - \mathbb{T}_\mathbb{C} - C_\eta^t$.

- *Step3*: (Decide the negotiation operation) Decide one kind of unattempted negotiation operations randomly for the current C_η^t. If it is *"self-adjusting"*, go to *Step4*, else if it is *"swapping"*, go to *Step5*, else it is *"inserting"* and go to *Step6*. If there was no successful negotiation in the above three kinds of negotiation, let $\mathbb{T}_\mathbb{C} \leftarrow \mathbb{T}_\mathbb{C} + \{C_\eta^t\}$, $\mathbb{C}_\delta \leftarrow \Phi$ and go back to *Step2*.

- *Step4*: (Improve by *self-adjusting*) If $t \geq t^*$, go to *Step8*, else if $C_\eta^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step3*, else choose a module m_p randomly from $C_\eta^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$. In the case that there exists a module m_q ($m_q \in C_\eta^t - \mathbb{T}_\mathbb{M}$, $m_q \neq m_p$) satisfying *evaluation criteria(1)* by exchanging the positions of m_p and m_q, update C_η^t, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go to *Step7*. In the case that there does not exist m_q satisfying *evaluation criteria(1)*, let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step4*.

- *Step5*: (Improve by *swapping*) If $t \geq t^*$, go to *Step8*, else if $C_\eta^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step3*; else choose a module m_p randomly from $C_\eta^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$. In the case that there exists a module m_q ($m_q \in C_\delta^t$ and $C_\delta^t \in \mathbb{C}_\delta$) satisfying *evaluation criteria(1) and (2)* (when $C_\eta^{t-1} = C_\gamma$) or *evaluation criteria(2)-(4)* (when $C_\eta^{t-1} \neq C_\gamma$) by swapping m_p and m_q, let $C_\eta^t \leftarrow C_\eta^t + \{m_q\} - \{m_p\}$ and $C_\delta^t \leftarrow C_\delta^t + \{m_p\} - \{m_q\}$, and let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go to *Step7*. In the case of that there does not exist such a module satisfying *evaluation criteria(1) and (2)* or *evaluation criteria(2)-(4)*, let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step5*.

- *Step6*: (Improve by *inserting*) If $t \geq t^*$, go to *Step8*, else if $C_\eta^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step3*; else choose a module m_p randomly from $C_\eta^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$. In the case that there exists a module m_q ($m_q \in C_\delta^t$ and $C_\delta^t \in \mathbb{C}_\delta$) satisfying *evaluation criteria(1) and (2)* (when $C_\eta^{t-1} = C_\gamma$) or *evaluation criteria(2)-(4)* (when $C_\eta^{t-1} \neq C_\gamma$) by inserting m_q in the back position of m_p, let $C_\eta^t \leftarrow C_\eta^t + \{m_q\}$ and $C_\delta^t \leftarrow C_\delta^t - \{m_q\}$, and let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go to *Step7*. In the case that there does not exist such a module satisfying

evaluation criteria(1) and (2) or *evaluation criteria(2)-(4)*, let $\mathbb{T}_{\mathbb{M}} \leftarrow \mathbb{T}_{\mathbb{M}} + \{m_p\}$ and go back to *Step6*.

- *Step7*: (Reset local variable) Let $\mathbb{C}_\delta \leftarrow \Phi$. In the case of $C_\eta^{t-1} = C_\gamma$, let $\mathbb{T}_{\mathbb{C}} \leftarrow \Phi$ and go back to *Step1*. In the case of $C_\eta^{t-1} \neq C_\gamma$, (a) if the successful negotiation is derived by *self-adjusting*, let $\mathbb{C}_\delta \leftarrow \{C_\eta^t\}$, else let $\mathbb{C}_\delta \leftarrow \{C_\eta^t\} + \{C_\delta^t\}$, (b) $\mathbb{T}_{\mathbb{C}} \leftarrow \mathbb{T}_{\mathbb{C}} - \{C_r\}$, and (c) go back to *Step2*.

- *Step8*: (Judge end conditions) In the case of $\mathbb{C} = \mathbb{T}_{\mathbb{C}}$ or $t \geq t^*$, end the negotiation procedure.

Evaluation criterion:
1) $E(C_\eta^t) < E(C_\eta^{t-1})$
2) $E(C_\delta^{t-1}) = E(C_\gamma)$ && $E(C_\delta^t) \leq E(C_\gamma) \parallel E(C_\delta^{t-1}) \neq E(C_\gamma)$ && $E(C_\delta^t) < E(C_\gamma)$
3) $E(C_\eta^{t-1}) = E(C_\gamma)$ && $E(C_\eta^t) \leq E(C_\gamma) \parallel E(C_\eta^{t-1}) \neq E(C_\gamma)$ && $E(C_\eta^t) < E(C_\gamma)$
4) $E(C_\eta^t) + E(C_\delta^t) < E(C_\eta^{t-1}) + E(C_\delta^{t-1})$

Thus, $\mathbb{C}$ is the final solution. And the algorithm strategy is consistent with the five step focusing process of TOC as shown in Figure 3.6.

Finally, we summarize the essential correspondences between DBR and our proposing approach for managing activities occurring in general MAS.

If $C_\eta^t = C_\gamma$, the system constraint is attempted to be exploited directly; else $C_\eta^t \neq C_\gamma$, it is attempted to subordinate/synchronize everything else in order to exploit the system constraint indirectly later. While analogizing DBR discussed in the section 2.3.1 (pp.13),

1) C_γ is just the *drum*.

2) When $C_\eta^t = C_\gamma$, containers in $\mathbb{C}_\delta$ are ready to improve the constraint directly, so it is alike a *rope* tying from them to the constraint (Especially $\mathbb{C}_\delta$ is dynamic and intelligent because it only includes containers that can do cooperative negotiations with C_η^t most possibly at present).

3) To a CLP, the *buffer* is managed through controlling values of evaluation functions of all containers.

In particular, the essential in a CLP is to elevate the constraint by reducing the value of the constraint's evaluation function and subordinating else containers, while the *buffer* being curtailed.

As we mentioned, the core property of MAS might be its self-organized and/or distributed problem solving, but in the real world purely bottom-up management would be difficult. Thus, if we let MAS work in a coordinated fashion, minimum management technique that can allow and preserve the agent's autonomous abilities will be required. For this purpose, we introduce DBR in MAS for a CLP.

The original idea of DBR and/or TOC was proposed as a technique for managing chained activities such as sequential manufacturing processes and more general supply chains. With respect to this, a CLP formalized in MAS can be also regarded

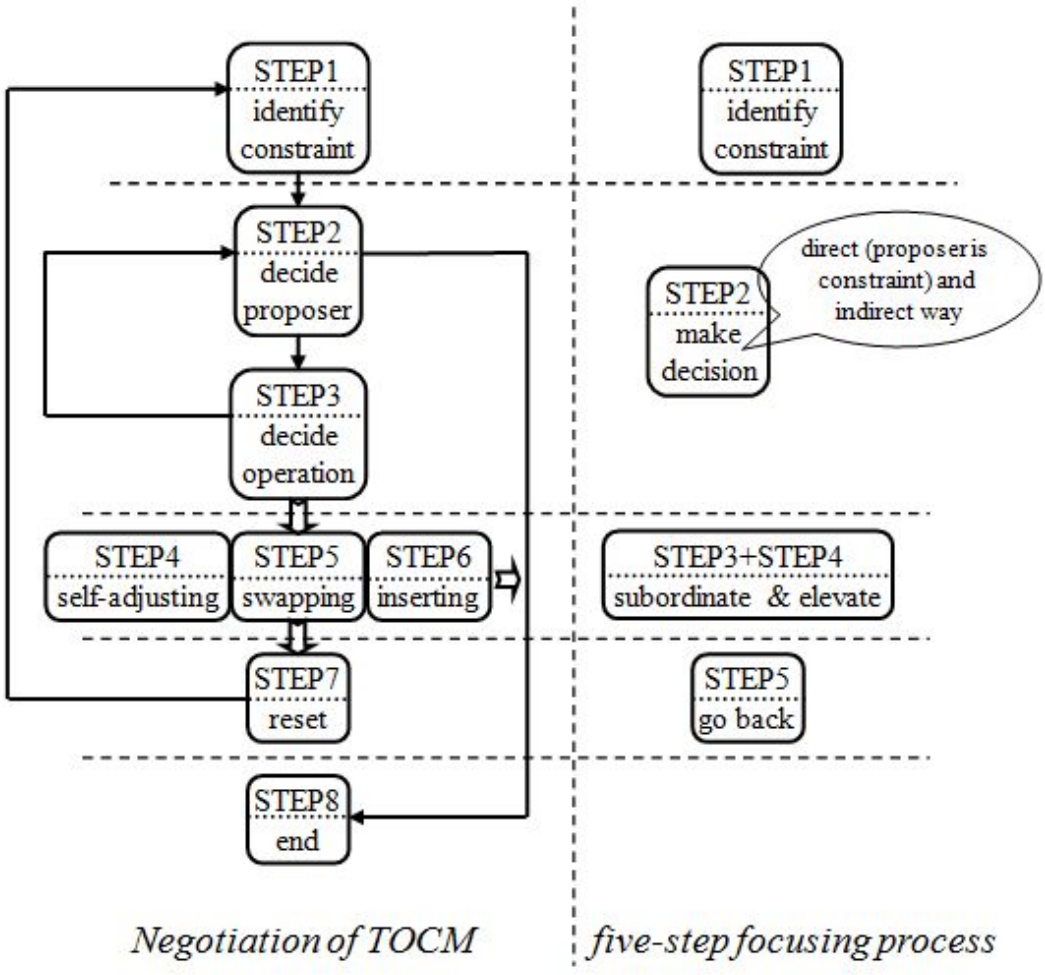


Figure 3.6: Corresponding between TOCM and Five-step Focusing Process

as an activity that consists of many negotiations among agents and those negotiations should be performed in some coordinated orders, i.e., not in a random order. Thus an idea of DBR can work for managing activities in MAS.

In the initial solution, the container C_γ having the maximum evaluation value is the constraint in DBR and activities of the other unconstrained containers should be synchronized with this constraint (i.e., a function of *drum*). In our approach, all negotiations are to elevate the constraint directly or indirectly and the evaluation criteria are decided by the situation of the constraint. In this sense, the difference of evaluation values between the best and the worst container in the initial solution can be regarded as the length of a *"rope"* that restricts the activities of unconstrained agents and let them synchronized with the constraint by allowing them to negotiate with each other for the trade of modules. As a result of this concessive negotiation, MAS may lead to finding another solution satisfying a more global optimality.

Herein, the expected difficulty those agents may encounter is to reach their consensus of exchanging the modules sacrificing the preservation of their evaluation value. If all possible negotiations are not successful, the problem solving will be stuck in the deadlock. In our approach, to avoid this deadlock situation, the idea of relaxation is introduced in the approach, which means the soft restrictions can be relaxed in a certain of extent. Through this relaxation, it becomes more possible for a constraint agent to find more partners of successful negotiations, thus the solution may be improved more. In this sense, this activity of relaxing soft restrictions for the constraint agent corresponds to a *buffer* management. The more the restrictions are relaxed, the more the *buffer* is increased, but this situation does not always lead to finding the global optimal solution, i.e., the bottleneck will be apt to move to other agent except the current constraint agent. On the other hand, if the restrictions are less to be relaxed, the *buffer* turns out to be small, and this may bring about the occurrences of deadlocks more frequently. Therefore, the *buffer* management for a constraint agent would be of importance for the management of MAS, which is included in the above approach.

3.4 Computation experiments

Some typical problems are listed in this research for examining its validity.

In Table 1, some basic properties of these problems are listed. These problems have large differences in packing order, cargo type, cargo dimension and cargo weight.

In this experiment,

1) t^* is initialized with 2×10^4.

2) $u_i \leftarrow 1.0$ $(1 \leq i \leq 5)$.

3) In Problem1, the *40FT* container (length: 11.9(m), width: 2.35(m), height: 2.35(m), carrying capacity: 26.67(t)) is used. And in other problems, the *20FT*

Table 3.1: Statistic properties of four problems

	Problem1	Problem2	Problem3	Problem4
N_o	1	1	5	5
N_{ct}	30	142	34	133
N_c	1003	910	1078	845
N_m	310	267	269	250
$L_{av}(mm)$	9.72×10^2	1.04×10^3	1.10×10^3	1.04×10^3
L_{sd}	5.12×10^{-1}	6.35×10^{-2}	2.08×10^{-1}	6.75×10^{-2}
$L_{min}(mm)$	870	870	970	870
$L_{max}(mm)$	1070	1144	1140	1140
$W_{av}(mm)$	8.47×10^2	8.74×10^2	9.23×10^2	8.74×10^2
W_{sd}	8.33×10^{-2}	4.69×10^{-2}	3.32×10^{-1}	4.91×10^{-2}
$W_{min}(mm)$	800	800	860	800
$W_{max}(mm)$	954	990	984	984
$H_{av}(mm)$	6.37×10^2	6.02×10^2	5.21×10^2	6.19×10^2
H_{sd}	3.39	1.77	1.92	1.88
$H_{min}(mm)$	166	166	186	166
$H_{max}(mm)$	700	700	700	700
$Wg_{av}(kg)$	1.84×10^2	2.46×10^2	2.80×10^2	2.29×10^2
Wg_{sd}	5.63×10^1	3.91×10^1	8.21×10^1	3.84×10^1
$Wg_{min}(kg)$	51	51	202	51
$Wg_{max}(kg)$	425	768	550	550

N_o: total number of packing orders; N_{ct}: total number of cargo types; N_c: total number of cargoes; N_m: total number of modules; L_{av}, L_{sd}, L_{min} and L_{max}: average, standard deviation, minimum and maximum of cargo lengths; else items are of cargo widths, cargo heights and cargo weights, respectively

container (length: 5.898(m), width: 2.35(m), height: 2.25(m), carrying capacity: 17.95(t)) is used.

4) The experiment is carried out on Microsoft Window XP. The CPU and memory of the used desktop computer is Intel Pentium-4 2.8 GH_z and 1.0 G., respectively. And the algorithm code is implemented with Visual C++.

The result of the negotiation is shown in Table 2. As discussed above, TOC is utilized in the proposed approach. The objective of the proposed approach as well as TOC is to elevate the system's constraint and get the biggest profit of the whole system. The achievement of the two objectives can be incarnated completely through statistics in Table 2. First, E_{max} has been reduced by 58%, 44%, 53% and 70% respectively and this means that the worst container has been improved greatly. Second, E_{av} has been reduced by 31%, 33%, 40% and 49% respectively and

Table 3.2: Situation of evaluations

	Problem1	Problem2	Problem3	Problem4
N_{con}	12	22	23	21
N_s	116	198	208	185
t_{max}	11935	11233	7952	8844
$Time(s)$	189	54	8	5
E_{av}^{0}	4.79×10^{-1}	3.54×10^{-1}	2.98×10^{-1}	4.72×10^{-1}
$E_{av}^{t_{max}}$	3.31×10^{-1}	2.36×10^{-1}	1.78×10^{-1}	2.39×10^{-1}
E_{sd}^{0}	4.73×10^{-1}	3.23×10^{-1}	3.11×10^{-1}	5.37×10^{-1}
$E_{sd}^{t_{max}}$	3.22×10^{-1}	2.13×10^{-1}	1.38×10^{-1}	1.86×10^{-1}
E_{max}^{0}	1.35	1.01	8.50×10^{-1}	1.34
$E_{max}^{t_{max}}$	5.68×10^{-1}	5.62×10^{-1}	4.02×10^{-1}	4.03×10^{-1}
E_{diff}^{0}	1.21	8.52×10^{-1}	7.16×10^{-1}	1.34
$E_{diff}^{t_{max}}$	5.37×10^{-1}	4.42×10^{-1}	2.73×10^{-1}	4.03×10^{-1}

N_{con}: total container number; N_s: number of successful iterations; t_{max}: total iteration number; $Time$(s): Computation time; E_{av}, E_{sd} and E_{max}: average, standard derivation and maximum of evaluation functions; E_{diff}: difference between the minimum and maximum of evaluation functions

this means that the solution has been improved greatly from the whole perspective. Anyway, the effectiveness of the final solution has been improved greatly.

Another special characteristics derived from TOC is to make agents of a system as synchronous as possible. This can be reflected through E_{sd} and E_{diff} and the smaller the value is, the more synchronous it is. In this experiment, E_{sd} has been reduced by 32%, 34%, 56% and 65% respectively and E_{diff} has been reduced by 55%, 48%, 61% and 70% respectively.

In additional, the evaluation variation of Problem3 is drawn in Figure 3.7 by recording successful negotiations and Figure 3.7(b) is the front part of Figure 3.7(a). Then some details of the proposed approach can be introduced as follows:

1) The basic trend of the evaluation of the constraint is decreasing. But the variations of the proposer and cooperator are random.

2) At the initial stage, generally the constraint of the system is easy to be improved and new one is identified iteratively.

3) If the constraint is fixed, which means the system runs into a deadlock situation, the average of evaluations is reduced and the whole system can also be improved because of successful negotiations of unconstrained agent.

4) The system deadlock situation is possible to be improved after successful negotiations of unconstrained agents (for example, it is at t=1765).

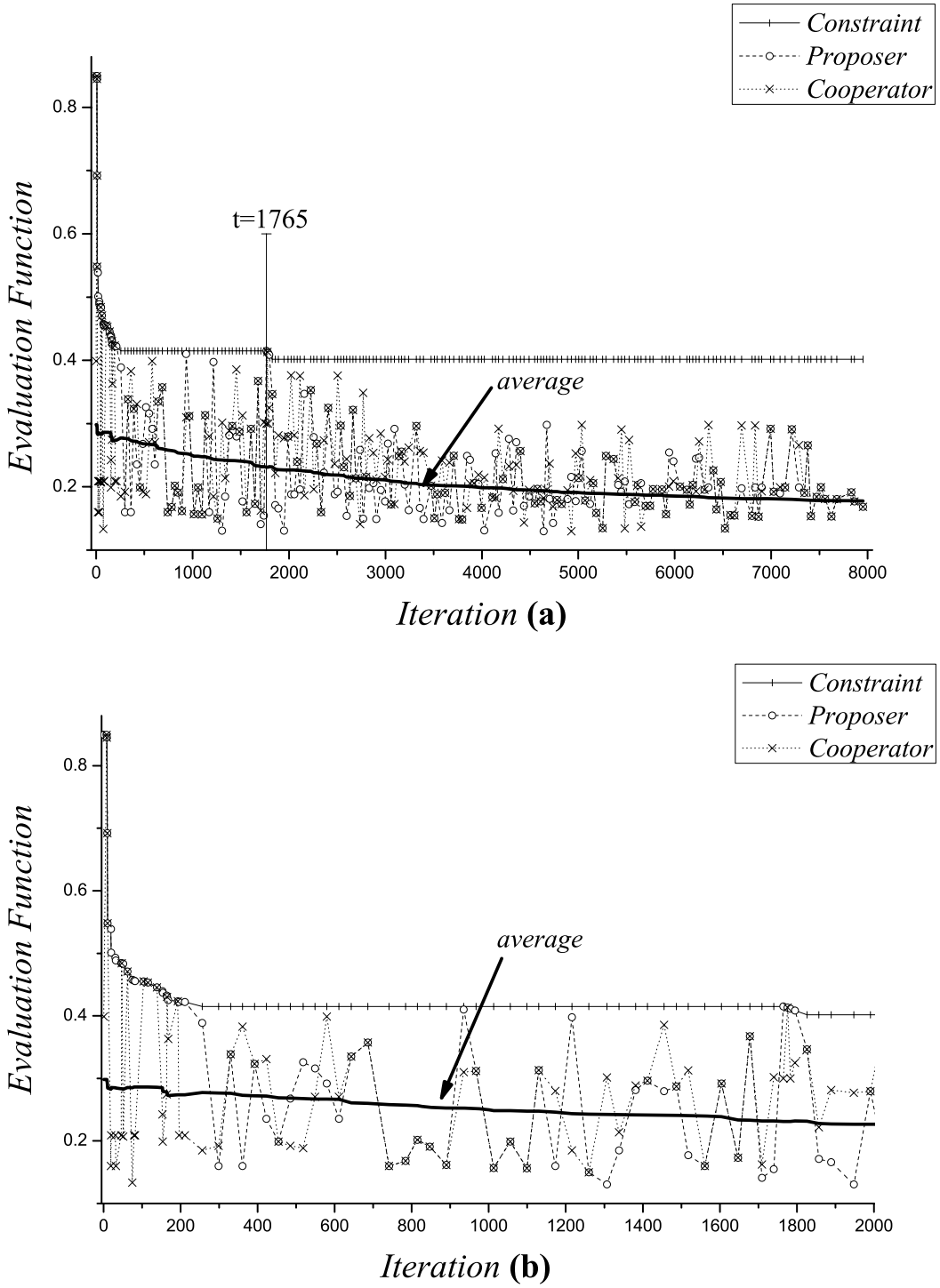


Figure 3.7: Variation of constraint, proposer, cooperator and average evaluation

Based upon the observation shown in Figure 3.7(b), we describe the dynamical characteristics of finding solutions that our approach realizes according to the TOC process. At first, the constraint is identified that has the worst evaluation function. In the graph, this is illustrated by the most lefthand plot initiating the entire iterations. In the following steps, MAS attempts to find better solution by individual agents' autonomous efforts without being supervised by any. However, such an autonomous problem solving without any control may sometimes go wrong failing in finding better global solutions. Therefore, we introduce an idea of *rope* that restricts the agents who are allowed to negotiate with others so that they could improve the solution as a whole. In other words, the negotiating activities of the unconstrained agents are controlled by the evaluation function value of the constraint agent, so the constraint plays a role of a pace-maker for other agents (i.e., in this sense, the constraint agent is regarded as a *drum* in TOC). During the initial stage of the iteration, this constraint agent is apt to change from one particular container agent to another. That is, a constraint agent is not fixed to a particular agent, but other agents may alternate a role of the constraint agent. This is observed during the initial 211 iterations of Figure 3.7(b), where the constraint agent plays a role of proposer at the same time, and if the negotiation is successful, the constraint agent move to another agent. What is characteristic here is that a constraint agent plays a role of pace-maker for all other unconstrained agents, and those latter agents attempt to make successful negotiations being constrained by the pace of the former as well as attaining the improvement of the total quality. From the 240th iteration, a constraint agent is fixed to a particular one without being alternated by the others. Even at this phase, the quality of the total solutions continue to be improved till the constraint is improved at t=1765. During this deadlock situation, one hand all negotiating activities among unconstrained agents are still supervised by the constraint; on the other hand, these negotiation activities are also to improve the constraint so the constraint is attempted to improve as soon as a successful negotiation among unconstrained agents. After some a suitable negotiation between unconstrained agents the constraint has a change to be improved at t=1765 and the system run out the current deadlock.

As mentioned at the end of the previous section, this control of relation corresponds to a *buffer* management, which affects on the frequency of occurrences of deadlocks in the following negotiation sessions. In the current approach, this *buffer* management is not done precisely, but is only done quite roughly (this is the reason that there are few changes to improve the constraint such as the right-hand of Figure 3.7(a)). The quality of solutions may be improved if we develop more precise ways of *buffer* managements.

3.5 Conclusion

The proposed TOCM is based on DBR and the five-step focusing process of TOC. It has been applied to dealing with the *two-row pattern* of a CLP and the process of loading container has been discussed in this research.

In TOCM, the restrictions required by users have been divided into hard and soft. Because it includes the cooperative negotiation strategy of autonomous agents, the real constraints of the system can be eliminated as fast as possible and agents are made more equilibrious after the cooperative negotiation. And it is easy to run out of a deadlock situation because of its negotiation strategy and the effect of an intelligent *rope*.

It has been proved that the *two-row pattern* of a CLP can be solved well by the proposed approach.

Chapter 4

Three-zone Buffer Management

4.1 Introduction

"Three-zone Buffer Management" (TZBM) is proposed for impoving "Theory of Constraints-based Management" (TOCM). And the *two-row pattern* of a container loading problem (CLP) will be discussed again in this chapter.

In section 4.2, the new problem definition and evaluation functions of the *two-row pattern* are introduced. Particularly, the new way is for adapting to the relaxation mechanism of TZBM.

Then the loading process is mentioned in the section 4.3.

The computation experiment is shown in the section 4.4. Especially, TZBM is compared with not only TOCM, but also two traditional optimization algorithms.

And finally the conclusion is in the section 4.5.

4.2 Definition of Container Loading Problem

Basic definitions and assumptions of the *two-row pattern* of a CLP in this chapter is same with ones in the previous chapter, please reference to paragraph 1-5 of the section 3.2 (pp. 21-22).

Restrictions and corresponding evaluation functions in this problem are repeated or redefined as follows:

Hard Restrictions:

- $\mathbf{H_1}$: The packing orders must obey FILO.
- $\mathbf{H_2}$: The total weight of modules loaded in a container cannot exceed the carrying capacity of the container, i.e.

$$\sum_{k=1}^{N_m} mg_k \leq CC \tag{4.1}$$

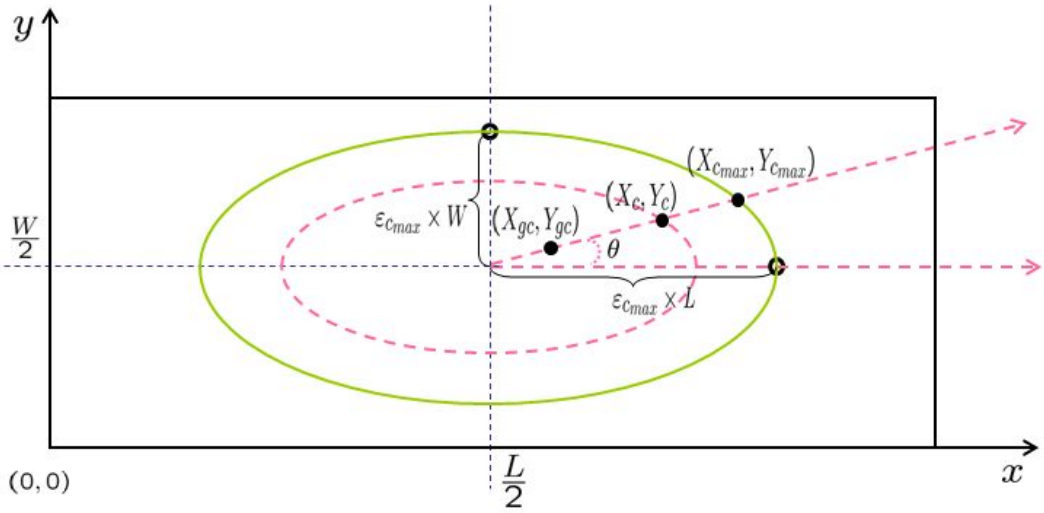


Figure 4.1: Soft restriction of gravity center (topview)

- **H$_3$**: Each module must be included completely within a container.

Soft Restrictions:

- **S$_1$**: If the gravity center of the container floor is (X_{gc}, Y_{gc}) and θ is the angle between x-axis and a radial linking the geometry center and the gravity center (seeing Figure 4.1), (X_{gc}, Y_{gc}) should be satisfied as :

$$\begin{cases} (X_{gc} - \frac{L}{2})^2 + (Y_{gc} - \frac{W}{2})^2 \leq (X_c - \frac{L}{2})^2 + (Y_c - \frac{W}{2})^2 \\ X_c = L/2 + \varepsilon_c \times L \times cos\theta \\ Y_c = W/2 + \varepsilon_c \times W \times sin\theta \end{cases} \tag{4.2}$$

  where ε_c is a relaxation parameter related to the gravity center and $\varepsilon_c \in [\varepsilon_{c_0}, \varepsilon_{c_{max}}]$ ($\varepsilon_{c_0} \leftarrow 0.125$ and $\varepsilon_{c_{max}} \leftarrow 0.25$ in this research according to the user's request).

- **S$_2$**: The balance between the left and right row is important for a *two-row pattern* of a CLP. The length of the left and right row as shown in Figure 4.2 should be satisfied as:

$$|L_{left} - L_{right}| \leq \varepsilon_b \times mw_{min} \tag{4.3}$$

  where mw_{min} is the minimum width of all modules. ε_b is a relaxation parameter related to the balance and $\varepsilon_b \in [\varepsilon_{b_0}, \varepsilon_{b_{max}}]$ ($\varepsilon_{b_0} \leftarrow 0.1$ and $\varepsilon_{b_{max}} \leftarrow 0.3$ in this research).

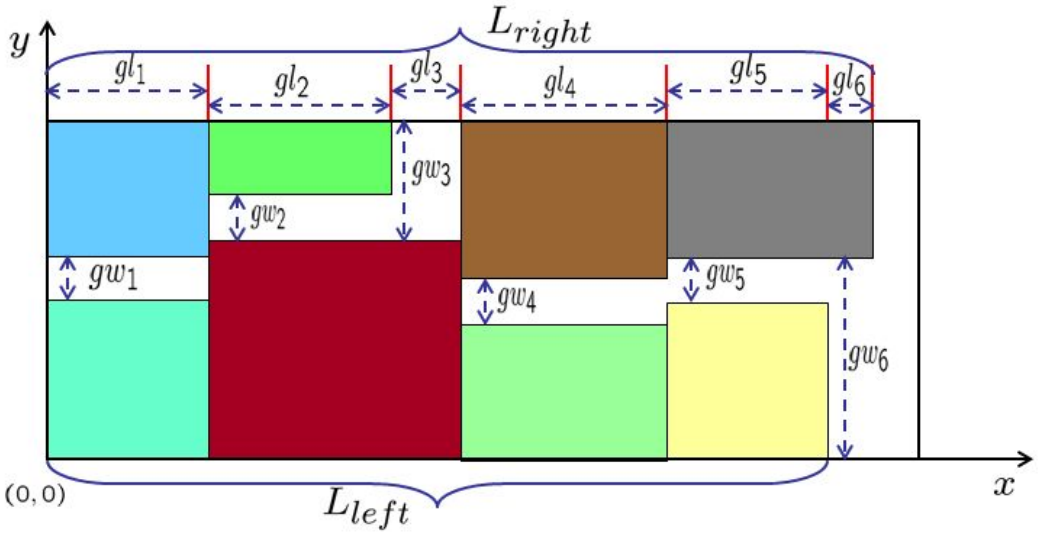


Figure 4.2: Inside six gaps (topview)

- **S$_3$**: a *two-row pattern* of a CLP requires to leave a certain distance between the two rows, which is called the regular gap written as gw_{reg} ($gw_{reg} \leftarrow 20cm$ in this research). The average wide of the gaps (seeing Figure 4.2) between the corresponding two modules belonging to different rows is defined as:

$$GW = \frac{\sum_{k=1}^{N_g}(|gw_k - gw_{reg}| \times gl_k)}{\sum_{k=1}^{N_g} gl_k} \tag{4.4}$$

where gl_k/gw_k is the length/width of the k-th gap, N_g is the total number of gaps in the current container. GW should be satisfied as:

$$GW \leq \varepsilon_g \times gw_{reg} \tag{4.5}$$

where ε_g is a relaxation parameter related to the gap wide and $\varepsilon_g \in [\varepsilon_{g_0}, \varepsilon_{g_{max}}]$ ($\varepsilon_{g_0} \leftarrow 0.25$ and $\varepsilon_{g_{max}} \leftarrow 0.5$ in this research).

- **S$_4$**: It should be avoided that the top cargo falls off from the higher module such as shown in Figure 4.3. The average of height differences of neighbor modules is defined as:

$$HD = \frac{\sum_{rw=left}^{right}\sum_{k=2}^{N_m^{rm}}|mh_{k-1} - mh_k|}{N_m^{left} + N_m^{right} - 2} \tag{4.6}$$

where N_m^{left}/N_m^{right} is the module number of the left/right row ($N_m^{left} + N_m^{right} > 2$). HD should be satisfied as:

$$HD \leq \varepsilon_h \times h_{min} \tag{4.7}$$

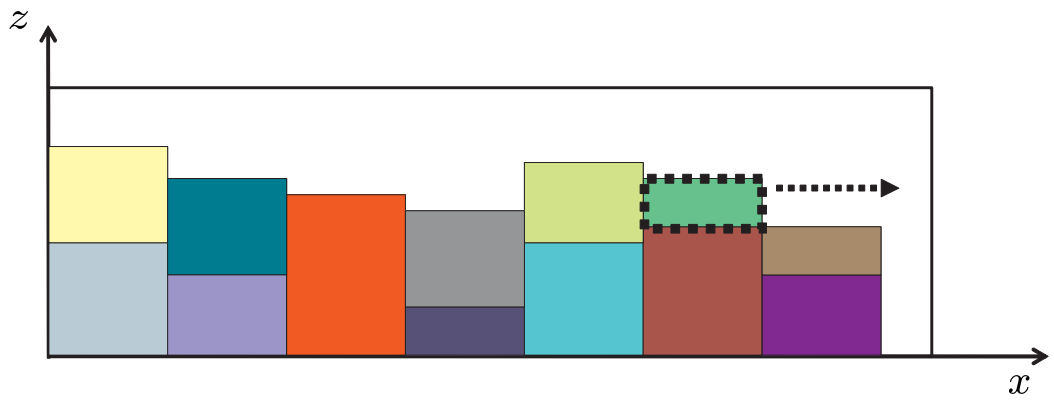


Figure 4.3: Unstable module (sideview)

where h_{min} is the minimum height of all cargoes, ε_h is a relaxation parameter related to the cargo stability and $\varepsilon_h \in [\varepsilon_{h_0}, \varepsilon_{h_{max}}]$ ($\varepsilon_{h_0} \leftarrow 0.1$ and $\varepsilon_{h_{max}} \leftarrow 0.3$ in this research).

Evaluation Functions:
According to above discussions, the sub-evaluation functions are defined as follows:

- $\mathbf{E_1}$: The sub-evaluation function for the gravity center is:

$$E_{s_1}(C) = \sqrt{\frac{(X_{gc} - \frac{L}{2})^2 + (Y_{gc} - \frac{W}{2})^2}{(X_{c_{max}} - \frac{L}{2})^2 + (Y_{c_{max}} - \frac{W}{2})^2}} \tag{4.8}$$

  $(X_{c_{max}}, Y_{c_{max}})$ is an intersection as shown in Figure 4.1 while ε_c being relaxed to the maximum.

- $\mathbf{E_2}$: The sub-evaluation function for the balance of the left and right row is:

$$E_{s_2}(C) = \frac{|L_{left} - L_{right}|}{\varepsilon_{b_{max}} \times mw_{min}} \tag{4.9}$$

- $\mathbf{E_3}$: The sub-evaluation function for the gap width is:

$$E_{s_3}(C) = \frac{GW}{\varepsilon_{g_{max}} \times gw_{reg}} \tag{4.10}$$

- $\mathbf{E_4}$: The sub-evaluation function for the cargo stability is defined as :

$$E_{s_4}(C) = \frac{HD}{\varepsilon_{h_{max}} \times h_{min}} \tag{4.11}$$

- $\mathbf{E_5}$: The sub-evaluation function for the floor occupied ratio is:

$$E_{s_5}(C) = 1 - \frac{\sum_{k=1}^{N_m} ml_k \times mw_k}{L \times W} \tag{4.12}$$

Based on these five sub-evaluation functions, the integrated evaluation function for a container can be described as:

$$E(C) = \sum_{i=1}^{5} u_i \times E_{s_i}(C) \tag{4.13}$$

where $u_i (i = 1, 2, ..., 5)$ are weighted parameters, which depend on how much the user prioritizes in relation with others.

Note that the bigger the E is the worse the loading situation is and the empty container does not join the negotiation.

4.3 Solving by TZBM

The loading process is divided into two parts.

The first part including "generating initial solution" and its addition "adjusting the heights of modules" is same with the one of TOCM, please reference to discussions of section 3.3.1 (pp. 25) and section 3.3.2 (pp. 26).

The second part is to improve the initial solution and introduced in the following sections.

4.3.1 Defining three zones

As mentioned above, loaded containers should satisfy many restrictions, which are usually interdependent and difficult to be satisfied simultaneously. In order to avoid the occurrence of violating situations as far as possible, an effective and feasible method called *restriction relaxation* is used in practice. For obtaining the optimal values of relaxed parameters and decreasing the useless relaxation, the approach TZBM is utilized.

In this *two-row pattern* of a CLP, four relaxing parameters ε_c, ε_b, ε_g and ε_h as shown in *soft restrictions* are defined to express the extent of relaxations. In the approach TZBM, all containers in $\mathbb{C}$ can be divided into three groups as follows:

- Green-zone ($\mathbb{GZ}$): if an agent (container) satisfies all non-relaxed restriction equations, i.e., $\varepsilon_i = \varepsilon_{i_0}$ $(i = \{c, b, g, h\})$, it is assigned to $\mathbb{GZ}$.

- Yellow-zone ($\mathbb{YZ}$): if an agent (container) satisfies all relaxed restrictions equations, i.e., $\varepsilon_{i_0} < \varepsilon_i \leq \varepsilon_{i_{max}}$ $(i = \{c, b, g, h\})$, it is assigned to $\mathbb{YZ}$.

- Red-zone ($\mathbb{RZ}$): the rest agents (containers) violate even the relaxed restrictions and they are assigned to $\mathbb{RZ}$.

A particular attention is focused on $\mathbb{RZ}$. Because agents in this zone violate the relaxed restrictions, one purpose of this research is to make $\mathbb{RZ}$ empty. The number of agents violating restrictions depends on the values of relaxed parameters. Usually, the minimum and maximum values of relaxed parameters are given by users. But the initial values of them can be set according to the computing time and the optimal values of them can be obtained by the algorithm. The large initial value can decrease the initial number of violating agents and decrease the computing time. In addition, the zones to be improved can be set to be $\mathbb{RZ}$ or $\mathbb{RZ} \cup \mathbb{YZ}$, which also depends on the computing time. If there is no enough time, the zones to be improved is set to be $\mathbb{RZ}$. In this research, the zones to be improved is set to be $\mathbb{RZ}$.

4.3.2 Improving initial solution

First, how to decide the weighted parameters defining the evaluation function is very significant. In order to allow the negotiation to all containers, the last loaded container C_{N_c} is dealt with differently from others. If its floor occupied ratio is low, all of its weighted parameters are set to be zero and it proposes no negotiation; else its u_1 and u_5 (corresponding to the gravity center and the floor occupied ratio) are set to be zero by default.

Second, some key variables of the negotiation procedure are listed as follows:

- t: the current iteration number;
- t^*: the maximum iteration number (depending on the given computing time);
- C_γ^t: the constrained container, which is from $\mathbb{RZ}$ with the maximum violation and proposes the negotiation conditions;
- C_δ^t: the cooperative container, which responds to the negotiation conditions proposed by C_γ^t;
- $\mathbb{T}_\mathbb{M}$: all modules having been tried in C_γ^t;
- $\mathbb{T}_\mathbb{C}$: the container list whose element was identified as C_γ^t ever but could not be elevated.

After C_γ^t being identified, three kinds of negotiation operations can be attempted in order to improve C_γ^t.

"Self-adjusting": the constrained container can be improved by changing the positions of its resources (modules).

"Swapping": the constrained container can be improved by exchanging its resources (modules) with other cooperative containers.

"Inserting": the constrained container can be improved by inserting resources (modules) from other cooperative containers.

If C_γ^t cannot be improved by any of the negotiations, which means that C_γ^t cannot be improved in the current situation, then it is added into $\mathbb{T}_\mathbb{C}$. If there is a successful negotiation carried out between C_γ^t and C_δ^t, C_γ^t and C_δ^t are assigned to their suitable zone(s) again according to their characteristics.

Finally the core steps of TZBM are described as follows:

- **Step1**: (Initialize parameters) Let $\varepsilon_i \leftarrow \varepsilon_{i_0}$, $\Delta\varepsilon_i \leftarrow \frac{\varepsilon_{imax}-\varepsilon_{i_0}}{10}$ $(i = \{c, b, g, h\})$; $t \leftarrow 0$, $\mathbb{T}_\mathbb{C} \leftarrow \Phi$ and $\mathbb{T}_\mathbb{M} \leftarrow \Phi$.

- **Step2**: (Divide into three zones) Divide all containers into three zones, $\mathbb{GZ}$, $\mathbb{YZ}$ and $\mathbb{RZ}$. If $\mathbb{RZ} = \Phi$, go to *Step9*.

- **Step3**: (Identify the system's constraint) If $\mathbb{RZ} = \mathbb{T}_\mathbb{C}$, go to *Step8*; else choose a container with the maximum value of sub-evaluation corresponding to unsatisfied soft restrictions from $\mathbb{RZ} - \mathbb{T}_\mathbb{C}$ as the constraint to be improved.

- **Step4**: (Decide the negotiation operation) Decide one kind of unattempted negotiation operations randomly for the current C_γ^t. If it is *"self-adjusting"*, go to *Step5*; else if it is *"swapping"*, go to *Step6*; else it is *"inserting"*, go to *Step7*. If these three kinds of negotiation operations have been attempted but C_γ^t can not be improved, let $\mathbb{T}_\mathbb{C} \leftarrow \mathbb{T}_\mathbb{C} + \{C_\gamma^t\}$ and go back to *Step3*.

- **Step5**: (Improve by *"self-adjusting"*) If $C_\gamma^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step4*. Otherwise, choose a module m_p randomly from $C_\gamma^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$.
  If there exists a module m_q ($m_q \in C_\gamma^t - \mathbb{T}_\mathbb{M}$, $m_q \neq m_p$) satisfying *evaluation criteria* (1)(2) by exchanging the positions of m_p and m_q, update C_γ^t and put it into the suitable zone. If $t < t^*$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step3*; else go to *Step9*.
  Otherwise in the case of that there does not exist such a module satisfying *evaluation criteria* (1)(2), let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step5*.

- **Step6**: (Improve by *"swapping"*) If $C_\gamma^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step4*. Otherwise, choose a module m_p randomly from $C_\gamma^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$.
  If there exists a module m_q ($m_q \in C_\delta^t$, $\delta \neq \gamma$, and belonging to the same group with m_p) satisfying *evaluation criteria* (1)(3)(4) by swapping m_p and m_q, let $C_\gamma^t \leftarrow C_\gamma^t + \{m_q\} - \{m_p\}$ and $C_\delta^t \leftarrow C_\delta^t + \{m_p\} - \{m_q\}$ and put C_γ^t and C_δ^t into the suitable zone(s) respectively. If $t < t^*$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step3*; else go to *Step9*.
  Otherwise in the case of that there does not exist such a module satisfying *evaluation criteria* (1)(3)(4), let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step6*.

- **Step7**: (Improve by *"inserting"*) If $C_\gamma^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step4*. Otherwise, choose a module m_p randomly from $C_\gamma^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$.
  If there exists a module m_q ($m_q \in C_\delta^t$, $\delta \neq \gamma$, and belonging to the same group with m_p) satisfying *evaluation criteria* (1)(3)(4) by inserting m_q in the back position of m_p, let $C_\gamma^t \leftarrow C_\gamma^t + \{m_q\}$ and $C_\delta^t \leftarrow C_\delta^t - \{m_q\}$ and put C_γ^t and C_δ^t into the suitable zone(s) respectively. If $t < t^*$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go to *Step3*; else go to *Step9*.
  Otherwise in the case of that there does not exist such a module satisfying *evaluation criteria* (1)(3)(4), let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step7*.

- **Step8**: (Relax parameters) If $\varepsilon_c < \varepsilon_{c_{max}}$, let $\varepsilon_i \leftarrow \varepsilon_i + \Delta\varepsilon_i$ ($i = \{c, b, g, h\}$), $\mathbb{T}_\mathbb{C} \leftarrow \Phi$ and go back to *Step2*. Otherwise, go to *Step9*.

- **Step9**: (Judge the end condition) In the case of no agent is in $\mathbb{RZ}$ or $\varepsilon_c \geq \varepsilon_{c_{max}}$ or $t \geq t^*$, end the negotiation procedure.

evaluation criteria:
1): no hard restriction is violated
2): $max\{u_i \times E_{s_i}(C_\gamma^t)\} < max\{u_i \times E_{s_i}(C_\gamma^{t-1})\}$; Or $max\{u_i \times E_{s_i}(C_\gamma^t)\} = max\{u_i \times E_{s_i}(C_\gamma^{t-1})\}$ and $E(C_\gamma^t) < E(C_\gamma^{t-1})(i = 1, 2, ..., 4)$
3): the size of $\mathbb{RZ}$ cannot be increased
4): $max\{u_i \times E_{s_i}(C_j^t)\} < max\{u_i \times E_{s_i}(C_j^{t-1})\}$; Or $max\{u_i \times E_{s_i}(C_j^t)\} = max\{u_i \times E_{s_i}(C_j^{t-1})\}$ and $E(C_j^t) < E(C_j^{t-1})(i = 1, 2, ..., 4,\ j = \gamma, \delta)$

Finally, $\mathbb{GZ} \cup \mathbb{YZ} \cup \mathbb{RZ}$ is the final solution.

In analogizing DBR discussed in the section 2.3.1 (pp. 13),

1) The constrained container C_γ^t takes a *drum* and proposes the negotiation conditions.

2) All cooperative containers obey the negotiation conditions and try to join the negotiations with C_γ^t, so it is alike a *rope* tying from C_γ^t to them.

3) For a CLP, the *buffer* is managed through controlling the parameters of relaxation and cooperative negotiations among agents.

In particular, it is essential in a CLP to elevate the constrained container by the cooperating behaviors in the relaxed situations. By doing so, the size of the Red-zone is reduced and the *buffer* is curtailed.

Here we summarize the essential correspondences again between DBR and our proposing approach for managing activities occurring in general MAS in detail again.

In the initial solution, the container C_γ^t violating the soft restrictions most strongly is the constraint in DBR and activities of the other cooperative containers should be synchronized with this constraint (i.e., a function of *drum*). In the proposed approach, the negotiation is carried out between the constrained agent and cooperative

agents. In this sense, the difference of how much soft restrictions being satisfied between the best and the worst container in the initial solution can be regarded as a *rope* that restricts the activities of cooperative agents and lets them synchronized with the constraint by allowing them to negotiate with each other for the trade of modules. If a negotiation is successful, i.e., both agents accept this trade in referring to their own evaluation functions, the violation degree of the constrained agent would be improved, while the change of the violation situation of the other is acceptable. As a result of this cooperative negotiation, MAS may find another solution satisfying a more global optimality.

Herein, the expected difficulty that those agents may encounter is to reach their consensus of exchanging the modules while sacrificing the preservation of their evaluation values. If all negotiations with the constrained agent are not successful, the problem solving will be stuck in the deadlock. In order to avoid the solving process falling down deadlock situations, a method called *restriction relaxation* used in practice is embedded in the proposed approach. By relaxing some a restriction, the solving process is changed to be flexible, more successful negotiations can be obtained and the constrained agents can be improved again. With the more restrictions being relaxed, deadlock situations are decreased in the solving process and the size of *buffer* is reduced gradually. However, the cooperative negotiation strategy can improve agents' violating states and reduce the size of *buffer* by adjusting constantly their evaluation.

4.4 Computation experiments

Some experiments are carried out for testify the validity of TZBM.

These experiments are carried out on Microsoft Window XP. The CPU and memory of the used desktop computer is Intel Pentium-4 2.8 GH_z and 1.0 G, respectively. And the algorithm code is implemented with Visual C++.

4.4.1 Comparing with traditional algorithms

Two kinds of optimization algorithms, "Algorithm LS" based on *Local Search* and "Algorithm TS" based on *Tabu Search*, are used to compare with TZBM. For comparing easily, three approaches use the same initial solution, the same definition of the searching technology (all containers can participate in the negotiation if they satisfy negotiation conditions proposed by C_γ^t), and the same judging criterion of successful negotiations (the first improved solution will be accepted) [24].

The core of "Algorithm LS" is to improve the current solution through searching a better solution from its neighborhood. The negotiation process can stop when the iteration reaches its maximum value. It is easy to run into a local optimum.

Table 4.1: Statistic properties of nine problems

	Problem1	Problem2	Problem3	Problem4	Problem5	Problem6	Problem7	Problem8	Problem9
N_o	1	1	3	5	1	1	3	2	1
N_{ct}	28	142	33	133	34	37	61	37	31
N_c	916	705	1908	1533	2662	4268	649	8391	1536
N_m	296	206	465	456	533	913	209	1015	385
$L_{av}(mm)$	1.00×10^3	1.04×10^3	1.10×10^3	1.04×10^3	1.02×10^3	1.08×10^3	1.08×10^3	1.08×10^3	1.05×10^3
L_{sd}	3.44×10^{-1}	4.58×10^{-2}	4.09×10^{-2}	6.75×10^{-2}	1.40×10^{-1}	2.29×10^{-1}	6.53×10^{-2}	2.28×10^{-1}	6.45×10^{-3}
$L_{min}(mm)$	870	870	970	870	870	980	870	980	870
$L_{max}(mm)$	1070	1144	1140	1140	1150	1150	1140	1150	1150
$W_{av}(mm)$	8.56×10^2	8.75×10^2	9.27×10^2	8.74×10^2	8.89×10^2	9.79×10^2	8.57×10^2	9.79×10^2	8.96×10^2
W_{sd}	6.17×10^{-1}	3.92×10^{-2}	4.57×10^{-1}	4.69×10^{-2}	1.96×10^{-1}	2.74×10^{-1}	3.41×10^{-3}	2.75×10^{-1}	4.34×10^{-2}
$W_{min}(mm)$	800	800	860	800	800	800	800	800	800
$W_{max}(mm)$	954	990	984	984	1150	1150	984	1150	1150
$H_{av}(mm)$	6.72×10^2	5.98×10^2	5.09×10^2	6.21×10^2	4.37×10^2	4.49×10^2	6.72×10^2	4.50×10^2	5.18×10^2
H_{sd}	2.78	1.33	3.25	1.87	6.76×10^{-1}	6.21×10^{-1}	5.15×10^{-1}	6.28×10^{-1}	5.58×10^{-1}
$H_{min}(mm)$	280	166	206	166	206	230	206	230	166
$H_{max}(mm)$	700	700	700	700	900	700	700	700	900
$Wg_{av}(kg)$	1.92×10^2	2.51×10^2	2.79×10^2	2.29×10^2	2.56×10^2	1.80×10^2	1.76×10^2	1.79×10^2	2.42×10^2
Wg_{sd}	4.05×10^1	3.01×10^1	1.62×10^1	3.80×10^1	1.40	5.13	2.79×10^1	4.95	1.40×10^1
$Wg_{min}(kg)$	114	51	202	51	104	107	59	107	51
$Wg_{max}(kg)$	425	768	550	550	658	292	520	292	658

N_o: total number of packing orders N_{ct}: total number of cargo types N_c: total number of cargoes N_m: total number of modules $L_{av}/L_{sd}\{L_{min}/L_{max}\}$ is the average, standard deviation, minimum and maximum of cargo lengths, etc.

"Algorithm TS" is proposed to improve the disadvantage of "Algorithm LS". A tabu list $\Gamma_{\mathbb{C}}$ storing searched solutions is defined to avoid repeating searches. When the iteration reaches its maximum value or $\mathbb{T}_{\mathbb{C}}$ equals with $\mathbb{C}$, the solving process is stopped.

The details of "Algorithm LS" and "Algorithm TS" are in the appendix of this chapter.

In this experiment, t^* is initialized with 5×10^4 and $u_i \leftarrow 1.0$ $(1 \leq i \leq 5)$ for TZBM.

In Table 4.1, some basic properties of some typical problems are listed. These problems have large differences in packing order, cargo type, cargo dimension and cargo weight.

Table 4.2 and Table 4.3 shows the number variations of containers in the Green-zone and in the Red-zone, respectively. The number is at different times during the problem solving process such as initial time, middle time (time=70 sec) and final time. It is found from Table 4.2 that in both the middle solution (at the middle time) and the final solution (at the final time), the number of containers in the Green-zone for TZBM is much more than those for "Algorithm LS" and "Algorithm TS" particularly in the final solution. Table 4.3 shows that in both the middle solution and the final solution, the number of containers in the Red-zone for TZBM is much less than those for other algorithms.

These results indicate that TZBM can not only improve containers violating restrictions but also make more containers satisfy the original restrictions. So TZBM is more effective for improving constrained containers violating restrictions and makes the whole system reach its goal faster than others.

Another important superiority of TZBM to other two algorithms is its "anytimeness". This idea was originally developed by Dean and Bobby [3], who referred to this idea as anytime computations, and Horvitz [16], who referred to this as flexible computations. In the flexible computation, the object-related value of the result (i.e., the value of the evaluation function of the solution) returned by the algorithm is a continuous function of the time spent in computation. The object-related value of a result is to be contrasted with the comprehensive value of a system's response to a given state. This means that the ideal algorithm should find a satisfactory solution as early as possible in computation, and should find an answer whose objective-related value increases as it is allocated additional computation time. Computation can be interrupted at any point, hence we should take account both of how good the solution is good enough with respect to its object-related value and of when that solution is obtained. This is the definition of comprehensive value. For the logistics problem, this characteristic of flexible computation is of great importance. The situation surrounding the real logistics problem may be changeable and dynamic due to the drifting market conditions, arrivals of unexpected express jobs, etc. Thus, a CLP should be flexible. Even though the algorithm found the better solution with greater object-related value, that would not be "bounded rational" if it takes longer

Table 4.2: Number of containers in Green-zone

Problem	Total number	Cont. number at initial time	Cont. number at time=70 sec			Cont. number at final time			
			TZBM	LS	TS	*Time*(*sec*)	TZBM	LS	TS
1	24	0	15	3	1	86	12	3	2
2	17	0	8	0	2	88	9	0	2
3	41	1	2	2	2	91	2	2	2
4	37	0	9	0	0	82	9	0	0
5	46	2	9	8	3	156	33	15	12
6	84	1	70	6	10	159	77	10	22
7	17	0	2	2	1	100	2	2	1
8	101	1	94	1	11	82	99	1	11
9	32	0	7	10	9	131	16	14	12

Cont. : container

Table 4.3: Number of containers in Red-zone

Problem	Total number	Cont. number at initial time	Cont. number at time=70 sec				Cont. number at final time			
			$(\varepsilon_i - \varepsilon_{i_0})/\Delta\varepsilon_i$	TZBM	LS	TS	$(\varepsilon_i - \varepsilon_{i_0})/\Delta\varepsilon_i$	TZBM	LS	TS
1	24	24	0	9	21	23	7	0	4	9
2	17	17	0	9	17	15	10	1	3	3
3	41	40	2	32	35	36	10	3	21	24
4	37	37	1	21	35	33	10	4	7	8
5	46	44	0	37	38	43	7	0	12	14
6	84	83	0	14	78	74	10	1	21	4
7	17	17	0	15	15	16	10	1	4	4
8	101	100	1	3	98	87	10	1	86	14
9	32	32	0	25	22	23	10	1	11	12

Cont. : container

Table 4.4: Maximum value and Difference of evaluation function

Problem	Initial max. evaluation	Final max. evaluation			Initial difference	Final difference		
		TZBM	LS	TS		TZBM	LS	TS
1	4.067	1.666	1.988	2.113	2.982	0.957	1.219	1.191
2	6.333	3.328	4.154	4.154	5.005	2.361	3.310	3.304
3	6.344	3.306	3.781	3.390	5.604	2.368	3.314	2.924
4	7.598	3.233	5.286	3.302	6.392	2.524	4.440	2.462
5	6.876	3.181	3.524	3.334	6.137	2.949	3.292	3.103
6	6.687	2.862	3.319	2.988	5.748	2.274	2.731	2.400
7	7.520	2.764	2.998	2.773	6.149	2.009	2.931	2.040
8	5.938	2.302	2.922	2.391	5.938	2.302	2.922	2.391
9	7.424	2.931	3.708	3.255	6.179	2.183	2.977	3.670

Maximum value of evaluation function means the one of the worst container
Difference of evaluation function means the one between the worst and best container

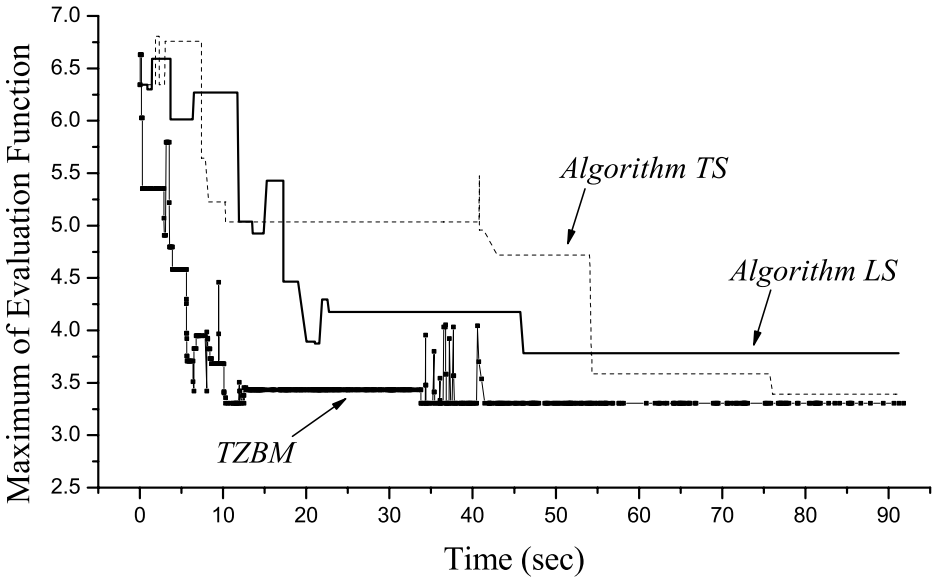


Figure 4.4: Maximum value of evaluation function of containers in problem 3

time to derive the solution. With respect to this, TZBM is superior to the other two algorithms as explicated in the results shown in Table II and Table III.

Table 4.4 shows the maximum value of evaluation functions of all containers and difference of evaluation functions between the worst and best containers. It is found from Table 4.4 that by comparing the initial solution, these three approaches have large improvement in the evaluation function of the worst container and the difference between the worst and best container. Particularly, TZBM is better than others.

For more details, the variation of the evaluation function of the worst agent is shown in Figure 4.4 and the difference of evaluation functions between the worst and best agents is shown in Figure 4.5 for problem 3. It is found from both Figure 4.4 and Figure 4.5 that the results of TZBM is better than others, which indicates that TZBM does not only focus its attention on the constrained agent but also reduce the difference among agent so that all agents in the system cooperate each other to reach their goals, i.e., TZBM is good at synchronizing agents.

The variation of zone sizes in problem 3 is shown in Figure 4.6. At the initial period, the agents in the Red-zone are difficult to be improved under the strict restrictions. After parameters being relaxed step by step, since the restrictions are relaxed and the solving space is changed, more successful negotiations among agents are obtained and zone sizes vary a lot. The size of the Red-zone decreases quickly,

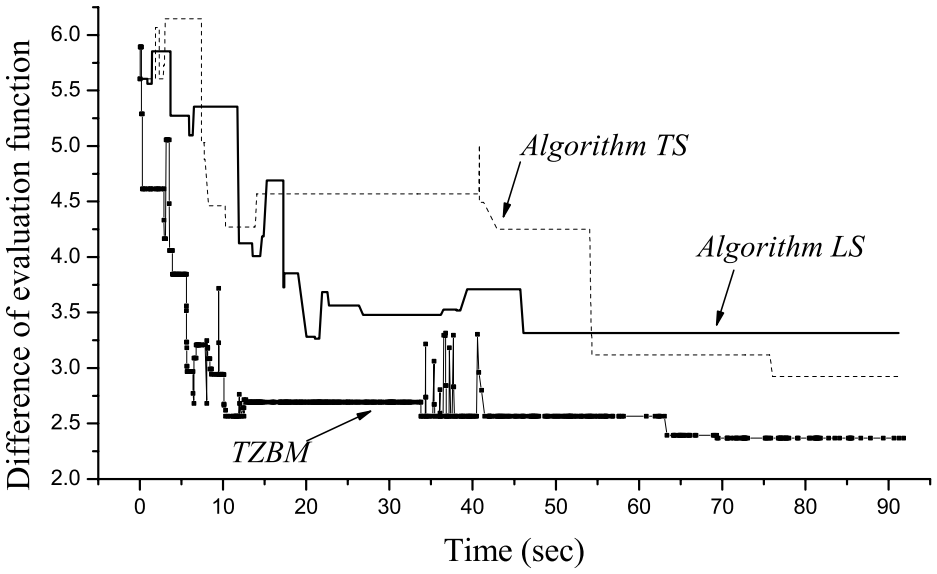


Figure 4.5: Difference of evaluation function between the worst and best container in problem 3

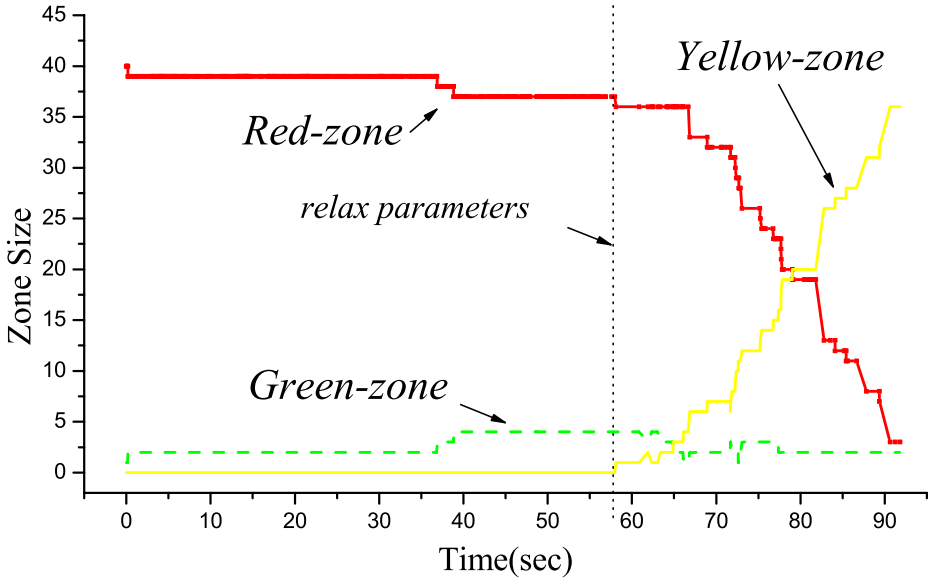


Figure 4.6: Variation of three zones in problem 3

Table 4.5: Number distribution of three zones of TZBM and TOCM

Problem	TZBM				TOCM		
	$(\varepsilon_i - \varepsilon_{i_0})/\Delta\varepsilon_i$	Green	Yellow	Red	Green	Yellow	Red
1	7	0	12	0	2	0	10
2	10	10	11	1	0	7	15
3	10	3	18	2	0	5	18
4	10	3	16	2	1	4	16

the size of the Yellow-zone increases accordingly and the Green-zone changes randomly.

As a matter of fact, TZBM can reduce restriction violations greatly and synchronize all agents as much as possible. And TZBM including the idea TZBM is of flexibility that a human being originally has.

4.4.2 Comparing with TOCM

In this experiment, used problems are same with ones in the section 3.4 (pp. 32). And t^* is initialized with 5×10^4 and $u_i \leftarrow 1.0$ $(1 \leq i \leq 5)$ in both TOCM and TZBM.

The computed results are shown in Table 4.5 (using zone dividing rule of TZBM). It is obvious that TZBM is better than TOCM at reducing the size of the Red-zone and violations to soft restrictions.

4.5 Conclusion

The proposed approach TZBM in this research is based on DBR and the five-step focusing process of TOC. It has been applied to solve the *two-row pattern* of a CLP and the process of loading container has been discussed in this research. The restrictions required by users have been divided into hard and soft, and the relaxing parameters have been defined for controlling the relaxing extent of the soft restrictions simultaneously. Because it includes the cooperative negotiation strategy of autonomous agents and the restriction relaxation used by the human being, it can eliminate the real constraints of the system and has the flexibility like a human being.

As discussed in above sections, the proposed approach is good at improving the system's constraints and increasing the system efficiency because of the cooperating behaviors done by other autonomous agents. It is less frequent to trap in a deadlock because of the appropriating relaxation strategy. It can obtain a better solution at any stopping computing time because it focuses its attention on always solving the

constrained agents. It has a high degree of flexibility and diversity based on its searching technology and negotiating strategy. It has been proved that the *two-row pattern* of a CLP can be solved well by the proposed approach.

In the real world, there are other complicated problems whose characteristics are alike to the ones of the CLP. That is, they are of multiple objectives and of multiple restrictions. These restrictions are always self-contradictory and difficult to be satisfied simultaneously. And it is necessary to get the goal of the whole system. Our proposing TZBM can be used to solve this kind of problems.

APPENDIX I

Local Search

Variables:

t: the current iteration number;
t^*: the maximum iteration number;
C_γ^t: the proposing container of the negotiation operation;
C_δ^t: the cooperative container.
Algorithm LS is as follows:

- **Step1**: (Identify the proposing agent) Select a container randomly from $\mathbb{C}$ as C_γ^t.

- **Step2**: (Decide the negotiation operation) Decide one kind of negotiation operations randomly for C_γ^t. If it is *"self-adjusting"*, go to *Step3*; else if it is *"swapping"*, go to *Step4*; else if it is *"inserting"*, go to *Step5*.

- **Step3**: (Improve by *"self-adjusting"*) Choose a module m_p randomly from C_γ^t and let $t \leftarrow t + 1$. If there exists a module m_q ($m_q \in C_\gamma^t$ and $m_q \neq m_q$) satisfying *evaluation criteria* (1)(5) by exchanging the positions of m_p and m_q, update C_γ^t. If $t \geq t^*$, go to *Step6*; else go back to *Step1*.

- **Step4**: (Improve by *"swapping"*) Choose a module m_p randomly from C_γ^t and let $t \leftarrow t + 1$. If there exists a module m_q ($m_q \in C_\delta^t$, $\delta \neq \eta$ and belonging to the same group with m_p) satisfying *evaluation criteria* (1)(6) by swapping m_p and m_q, let $C_\gamma^t \leftarrow C_\gamma^t + \{m_q\} - \{m_p\}$ and $C_\delta^t \leftarrow C_\delta^t + \{m_p\} - \{m_q\}$. If $t \geq t^*$, go to *Step6*; else go back to *Step1*.

- **Step5**: (Improve by *"inserting"*) Choose a module m_p randomly from C_γ^t and let $t \leftarrow t + 1$. If there exists a module m_q ($m_q \in C_\delta^t$, $\delta \neq \eta$ and belonging to the same group with m_p) satisfying *evaluation criteria* (1)(6) by inserting m_q in the back position of m_p, let $C_\gamma^t \leftarrow C_\gamma^t + \{m_q\}$ and $C_\delta^t \leftarrow C_\delta^t - \{m_q\}$. If $t \geq t^*$, go to *Step6*; else go back to *Step1*.

- **Step6**: (Judge the end condition) In the case $t \geq t^*$, end the negotiation procedure.

evaluation criteria:
5): $E(C_\gamma^t) < E(C_\gamma^{t-1})$
6): $E(C_\gamma^t) + E(C_\delta^t) < E(C_\gamma^{t-1}) + E(C_\delta^{t-1})$

APPENDIX II

Tabu Search

Variables:

t: the current iteration number;
t^*: the maximum iteration number;
C_γ^t: the proposing container of the negotiation operation;
C_δ^t: the cooperative container;
$\mathbb{T}_\mathbb{M}$: all modules having been tried in C_γ^t;
$\mathbb{T}_\mathbb{C}$: the container list whose element was identified as C_γ^t ever but could not be elevated;
$\Gamma_\mathbb{C}$: the tabu list for saving last s moves ($s = 1 \times 10^3$ in this research) ;
ν: the experiential value to prevent too much deterioration ($\nu \leftarrow 0.03$ in this research)

Algorithm TS is as follows:

- **Step1**: (Identify the proposing agent) If $\mathbb{C} = \mathbb{T}_\mathbb{C}$, go to *Step6*; else select a container randomly from $\mathbb{C} - \mathbb{T}_\mathbb{C}$ as C_γ^t.

- **Step2**: (Decide the negotiation operation.) Decide one kind of unattempted negotiation operations randomly for the current C_γ^t. If it is *"self-adjusting"*, go to *Step3*; else if it is *"swapping"*, go to *Step4*; else if it is *"inserting"*, go to *Step5*. If these three kinds of negotiation operation have been attempted but C_γ^t can not be improved, let $\mathbb{T}_\mathbb{C} \leftarrow \mathbb{T}_\mathbb{C} + \{C_\gamma^t\}$ and go back to *Step1*.

- **Step3**: (Improve by *"self-adjusting"*) If $C_\gamma^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step2*. Otherwise, choose a module m_p randomly from $C_\gamma^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$. If there exists a module m_q ($m_q \in C_\gamma^t - \mathbb{T}_\mathbb{M}$, $m_q \neq m_p$) satisfying *evaluation criteria* (1)(5)(7) by exchanging the positions of m_p and m_q, update C_γ^t; Else if there is a set of modules satisfying *evaluation criteria* (1)(7)(8), choose m_q that can make C_γ^t having the minimum value of the evaluation function in the set and then update C_γ^t.
  In the case that there exists such a module satisfying *evaluation criteria* (1)(7)(8) at least, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and $\Gamma_\mathbb{C} \leftarrow \Gamma_\mathbb{C} + \mathbb{C}$. If $t < t^*$, go back to *Step1*; else go to *Step6*. On the contrary (in the case of no module satisfying *evaluation criteria* (1)(7)(8)), let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step3*.

- **Step4**: (Improve by *"swapping"*) If $C_\gamma^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step2*. Otherwise, choose a module m_p randomly from $C_\gamma^t - \mathbb{T}_\mathbb{M}$ and let $t \leftarrow t + 1$.
  If there exists a module m_q satisfying *evaluation criteria* (1)(6)(7) and by swapping m_p and m_q, update C_γ^t and C_δ^t; Else if there is a set of modules satisfying *evaluation criteria* (1)(7)(9), choose m_q that can make C_γ^t and C_δ^t have

the minimum sum value of evaluation functions in the set and then update C_γ^t and C_δ^t.
In the case that there exists such a module satisfying *evaluation criteria* (1)(7)(9) at least, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and $\Gamma_\mathbb{C} \leftarrow \Gamma_\mathbb{C} + \mathbb{C}$. If $t < t^*$, go back to *Step1*; else go to *Step6*. On the contrary (in the case of no module satisfying *evaluation criteria* (1)(7)(9)), let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step4*.

- **Step5**: (Improve by *"inserting"*) If $C_\gamma^t = \mathbb{T}_\mathbb{M}$, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and go back to *Step2*. Otherwise, choose a module m_p randomly from $(C_\gamma^t - \mathbb{T}_\mathbb{M})$ and let $t \leftarrow t + 1$.
  If there exists a module m_q satisfying *evaluation criteria* (1)(6)(7) by inserting m_q in the back of m_p, update C_γ^t and C_δ^t; Else if there is a set modules satisfying *evaluation criteria* (1)(7)(9), choose m_q that can make C_γ^t and C_δ^t have the minimum sum value of evaluation functions in the set and then update C_γ^t and C_δ^t.
  In the case that there exists such a module satisfying *evaluation criteria* (1)(7)(9) at least, let $\mathbb{T}_\mathbb{M} \leftarrow \Phi$ and $\Gamma_\mathbb{C} \leftarrow \Gamma_\mathbb{C} + \mathbb{C}$. If $t < t^*$, go back to *Step1*; else go to *Step6*. On the contrary (in the case of no module satisfying *evaluation criteria* (1)(7)(9)), let $\mathbb{T}_\mathbb{M} \leftarrow \mathbb{T}_\mathbb{M} + \{m_p\}$ and go back to *Step5*.

- **Step6**: (Judge the end condition) In the case of $\mathbb{C} = \mathbb{T}_\mathbb{C}$ or $t \geq t^*$, the negotiation process ends.

evaluation criteria:
7): $\mathbb{C} \notin \Gamma_\mathbb{C}$
8): $E(C_\gamma^t) - E(C_\gamma^{t-1}) < \nu \times E(C_\gamma^{t-1})$
9): $E(C_\gamma^t) + E(C_\delta^t) - E(C_\gamma^{t-1}) - E(C_\delta^{t-1}) < \nu \times (E(C_\gamma^{t-1}) + E(C_\delta^{t-1}))$

Finally, one solution in $\Gamma_\mathbb{C}$, which is with the minimum sum value of evaluation functions of all containers, is the final solution.

Chapter 5

Interaction-based Knowledge Acquiring Framework

5.1 Introduction

Both "Theory of Constraints-based Management" (TOCM) and "Three-zone Buffer Management" (TZBM) are approaches based on distributed and cooperative systems, but only computer agents are embedded in them. In this chapter, an approach including a human expert agent is discussed for improving the intelligence of solving *constraint satisfaction problem*s (CSPs) in the real world.

In the next section 5.2, the architecture of the proposed approach called "Interaction-based Knowledge Acquiring Framework" (IKAF) is presented in details. Then, it is applied to solve another general issue of the container loading problem (CLP), *block-building pattern*, in which cargoes are *extreme heterogeneous*, to illustrate how the framework does work.

Both the problem definition and concrete solution will be presented in the section 5.3.

In the section 5.4, the solving process of the *block-building pattern* is demonstrated as the validation of the application.

In the last section 5.5, conclusions are discussed.

5.2 Architecture

The architecture of IKAF is shown in Figure 5.1. Wherein, "Expert" is a human expert of the involved domain. "Problem" is an instance to be solved of the domain. "Result" is a result of "Problem" computed by the problem solver. "Example" is revised "Result" by the expert and can be analyzed by the EBL mechanism.

The Problem-Solver produces a heuristic search tree, encapsulating all child branching decisions as well as the final result. In order to solve problems in a

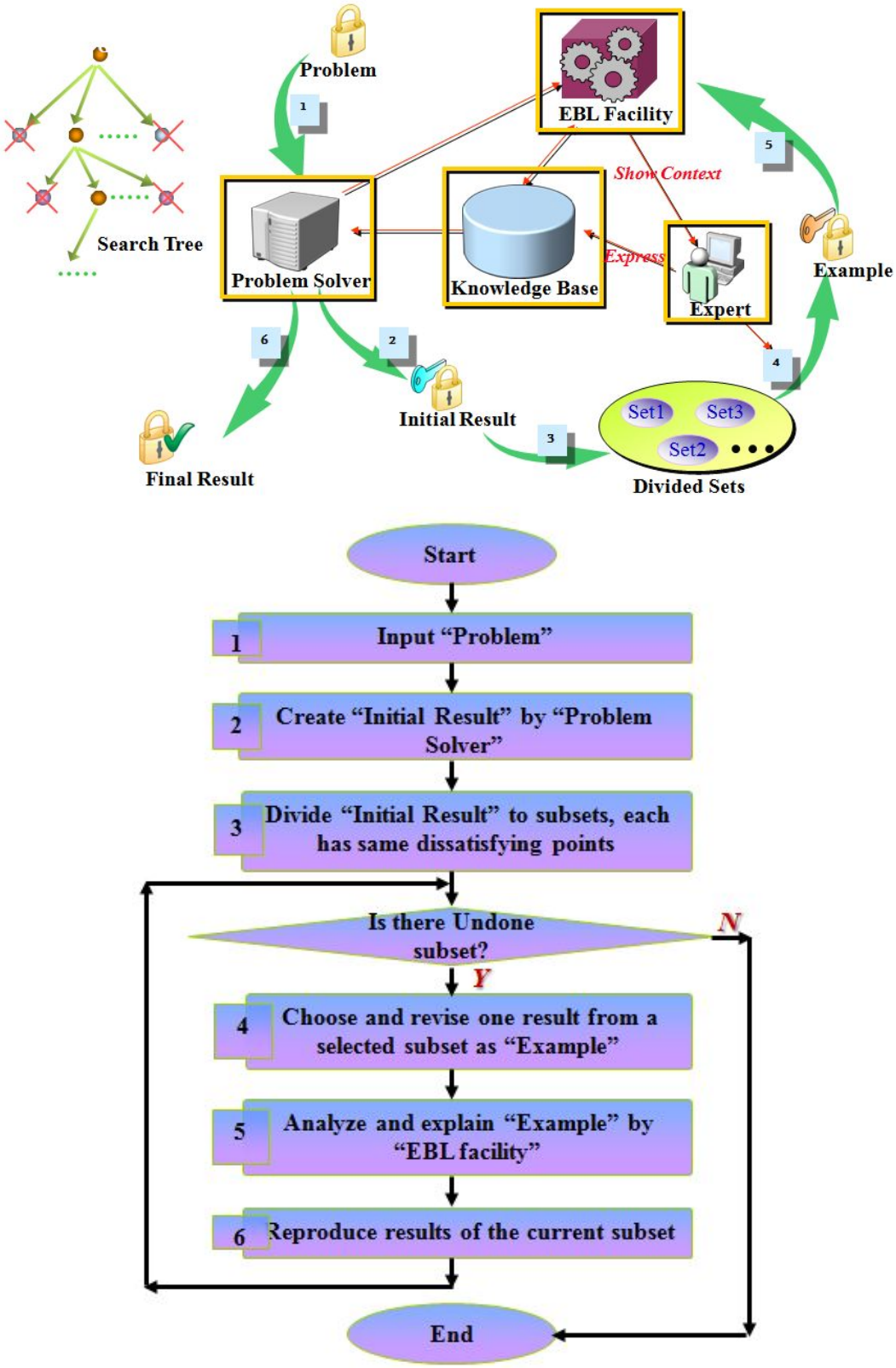


Figure 5.1: Interaction-based Knowledge Acquiring Framework

particular domain, the system must be given a specification of that domain, which consists of a set of operators and inference rules. Operators correspond to external actions with consequences in the world. Each operator has a precondition expression that must be satisfied before the operator can be applied, and a list of effects that describes how the application of the operator changes the current state of the world. Inference rules simply increase explicit knowledge about the current state [22].

The knowledge base stores the domain knowledge and control rules. The domain knowledge describes the relevant aspect of the problem solver in addition to a theory of the domain, which includes operators and inference rules mentioned in the above. In our framework, controls rules describe how to solve problems in the domain. According to Dorothy Leonard's discussion knowledge exists on a spectrum; at one extreme it is almost completely tacit and at the other end knowledge is almost completely explicit. Most knowledge exists in between the extremes [18]. So, control rules are expressed in the same formation but are divided into three types: *Gcr* are *general* control rules, which are common ones to all instances without special preconditions and need not be revised over time. Their definitions are logically rigorous and derived as the expert's common expertise. *Scr* are *soft* control rules, which are description of tacit knowledge and used in a more flexible way than the former. Different from the former, they cannot be guaranteed to be logically rigorous, and may include incomplete or contradictory rules. *Rcr* are *reinforced* control rule, which are converted from *Scr* through the EBL mechanism or added directly by "Expert" in our system (*Rcr* are always generalized as *Rcr'* by the function of EBL).

The EBL component has the key function in our system for preserving participation in the problem solving and interactions of an expert user with the system. This function for acquiring *Rcr* from *Scr* through analyzing and explaining the expert's demonstrating example of modification. The explanation is constructed from an axiomatized theory taking account both of knowledge base and relevant aspects of the problem solver's architecture. Moreover, this component also takes on the work to check the validation of control rules and to manage to preserve the consistency of the collection of evolvable and accumulated knowledge. In this way, the core functions of the EBL component is to check the consistency of the human experts' demonstrations in terms of the knowledge preserved within the system so far, and to give the appropriate advice to the human user when the system detects what is not consistent.

The other important function of the EBL component in our system is to distinguish between what is particular in a specific problem and what could be generalized and reusable in other problems as well. This function contributes with a great deal to distributing knowledge from one user to the others in managing and controlling the translation processes among three different knowledge sources of general control rules, soft control rules and reinforced control rules.

The working process of our system is as follows:

Step1, after getting a batch of instances the system tries to solve them only using *Gcr*. And initial results of all instances are recorded.

Step2, investigating into the whole obtained results, the system identifies a subset of results, which have common dissatisfying points.

Step3, the expert selects one from this subset and revises it into a more idealized one (i.e., an expert's demonstration of his/her tacit knowledge). At this time, the revised result also needs to be analyzed and confirmed by the expert in order to guarantee that it can be interpretable by the EBL mechanism.

Step4, the EBL component attempts to construct search trees recursively using not only *Gcr* but also *Scr* while parts of *Scr* identified as useful are converted to more generalized ones *Rcr'* and added into knowledge base. If there are not enough rules to explain any child node of the search tree (i.e., sub-problem needed to be resolved for completing a search), the expert must supply new ones into *Scr* and restart the analysis procedure. This is very significant and feasible because as for the tacit knowledge it is quite difficult for the expert to verbalize and to mention explicitly and exhaustively. However, if the problem is focused to a particular point and given, the expert can show a solution without much difficulty by imagining the problem with its surrounding reality. At this time, the EBL component can check validity of any newly added control rule by testing in old related nodes because all calculated nodes are recorded in the system.

Step5, in this way, the system can get better results of the first subset computed by the problem solver using *Gcr* plus *Rcr'* again.

Through iterating the above steps from Step2 to Step5 recursively, the system can get all satisfying results of the whole instances step by step. At this time, different *Rcr* may be derived in each loop. We think that different subsets should be classified into to respective classes, each of which has respective *Rcr* to guarantee both efficiency and effectiveness of the problem solving in the consequent problem solving sessions.

The core of our system is to be able to manage the evolvable knowledge dynamically. This is the major departure from the conventional knowledge-based system like expert systems, in which knowledge does not change once it was built in the knowledge base. In our system, this is realized by maintaining the control-rule set consistent among the variations of problems that belong to the same domain. Actually, the system is trying to avoid the following three undesired situations when encountering instances of the same domain:

1)"Incompletion": the system cannot build explanations of the problem because of missing knowledge.

2)"Intractable": the system have enough knowledge to make an explanation of this, but not enough computing time is available.

3)"Inconsistence": the system derives inconsistent results based upon a current theory (e.g., caused by the existence of some default rules).

If the system encounters some of the above cases, it will ask a human expert to

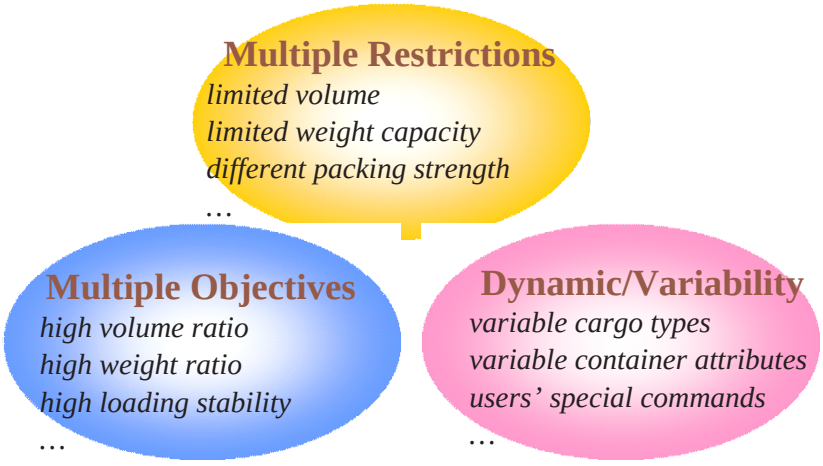


Figure 5.2: Difficulties of CLP with extreme heterogeneous cargoes

commit to resolve this conflict situations and to construct a robust knowledge base by redefining *operationality criterion*, that is a criteria for evaluating the acceptance of the knowledge to be captured by the system. This is done through a friendly human-machine interface.

5.3 Solution to Container Loading Problem

Let's review the situation of CLPs in the real world. In recent years, the software with artificial intelligence on allocating a truck or a container becomes very important and popular. The reasons are considered as follows: one reason is to decrease loading cost and the other is to increase loading speed. The skilled workers usually can allocate various large cargoes to a container effectively and fast, while an unskilled worker cannot do that. Moreover, companies usually make matters worse that the skilled workers are absent from work because of illness or some other things. When there are more than decades kinds of cargoes with small sizes, even the skilled workers cannot handle them under a limited time and the result is not satisfactory sometimes.

The CLP described in this chapter is the *block-building pattern*, in which many kinds of cargoes with different sizes, weights and numbers into containers need to be loaded and allocated with many complicated restrictions as shown in Figure 5.2. Moreover, multiple objectives and dynamic and variable characteristics are making the CLP as NP-hard problems.

According to the characteristics of problems, solving the *block-building pattern* of a CLP is generally divided into two operations. The first operation is to allocate cargoes into a series of blocks under satisfying the restrictions of loading blocks.

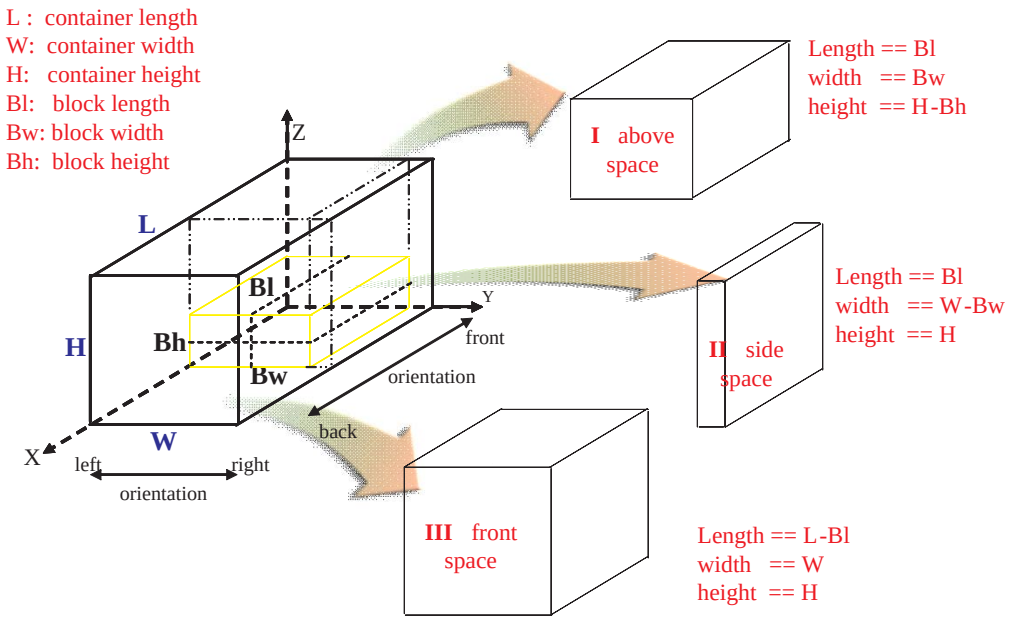


Figure 5.3: Block and Space split of block-building pattern of CLP

The second operation is to allocate the blocks into containers subject to the restrictions of allocating containers. In the followings, to simplify the procedure of the above CLP and to highlight the core functions of our proposing framework, we deal with this problem with the following assumptions.

5.3.1 Problem and Assumption

In our three-dimension container-loading problem, the following issues are assumed:

- Only single containers are considered. The container and cargos are rectangular and of known dimensions and weight.
- Cargos are loaded from left to right, from bottom to top and from front to back in the container (Especially the door side is regarded as the back). And on the three-dimension points a relation $<_p$ is defined so that for two points $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ the following conditions hold:
  $P_1 <_p P_2$, if $x_1 < x_2$;
  or if $x_1 == x_2$ and $y_1 < y_2$;
  of if $x_1 == x_2$ and $y_1 == y_2$ and $z_1 < z_2$.
- Each cargo is positioned parallel to the side walls of the container.
- Each cargo lies completely within the container. Cargo cannot be rotated vertically nor horizontally.

- Cargos can be loaded in any location in the container and with any other cargo.
- Cargo with weaker packing strength cannot be placed below the cargo with stronger packing strength.
- Container weight balance is not strict. The weight of a complete cargo must not exceed a given weight limit.

Formulation of the *block-building pattern* requires the introduction of two concepts. First, a special structure named "block", that is solid and cuboid, consists of same cargos. We load the cargo block instead of the individual single cargo one after another during the loading process. Sometimes one block includes only one cargo. Second, "space split" is also cited on how spaces are utilized. If one space is decided to match with one block, the block is placed at the left-bottom-front corner of the space, and then three new valid spaces, the above, the side and the front, are generated. "Block" and "Space split" are illustrated in Figure 5.3. This loading pattern is intuitive and easy to use. Although there is also another operation "space merge" which is to merge or recombine adjacent spaces to utilize space efficiently, we do not consider the space merging in this research.

Multiple capacity constraints are considered in this problem. First, no cargo is outside of the container or straddled the edges of the container. Second, the weight capacity of the container is abided strictly. Finally, the rotation and packing strength of the cargo must be considered.

Our goal is to load more cargos into the container to reduce the transport cost. Additionally, we must consider not only the volume ratio but also the weight ratio of the container. As a secondary criterion, we aim to make the operation of loading and unloading as easy as possible.

5.3.2 Computational Procedure

According to the description in section 5.3.1, the loading procedure shown in Figure 5.5 is developed to determine the container loading patterns that consist of the loading orientation (i.e. the *x*-*y*-*z* dimensions) of each cargo and its corresponding location (i.e. the reference point of loading space).

The pre-stored data for the system are the following two sets of data: the set $\mathbb{S}$ containing all usable spaces and the set $\mathbb{C}$ including all cargos to be loaded. Moreover, the data set $\mathbb{C}'$ is the subset of $\mathbb{C}$ and store the filtered elements of $\mathbb{C}$ which are possible to be used to make blocks (for example, when the packing strength of the block below the active space is weak, elements of $\mathbb{C}$ with strong packing strength are not possible to match with the active space and not included in $\mathbb{C}'$). The blocks made by elements of $\mathbb{C}'$ are stored in the set $\mathbb{B}$.

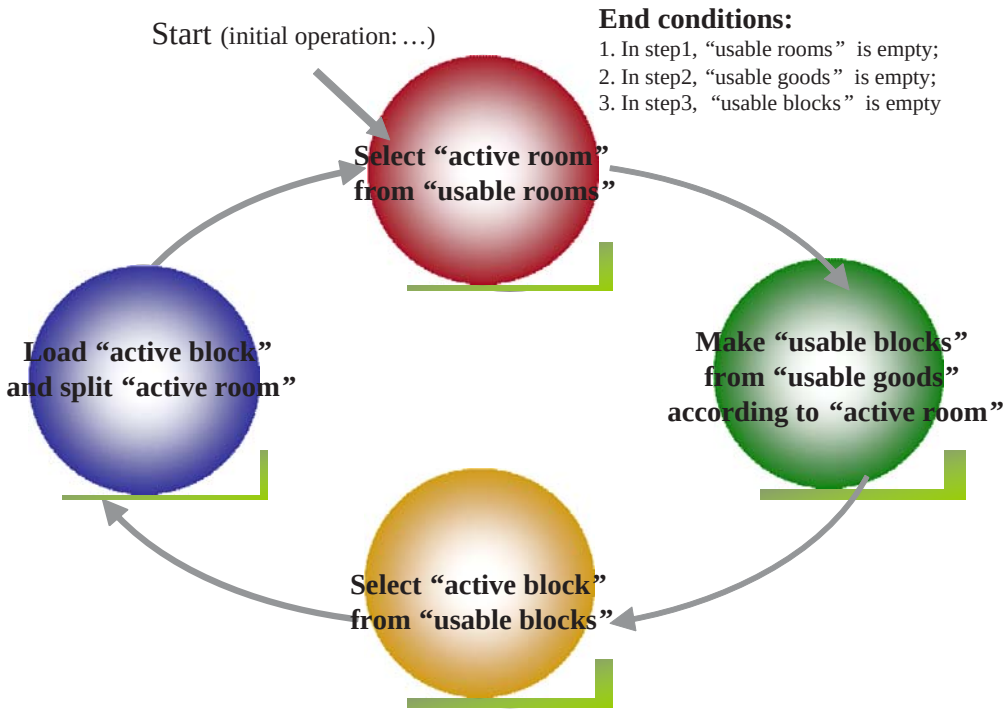


Figure 5.4: Four core decisions during single container loading

In step1, in starting a problem solving of a container-loading, the original information of the container and cargos are read from the above data sets. There stored an initial space in $\mathbb{S}$, whose size is the same with the inner size of the container initially. In step2, one space is selected from $\mathbb{S}$, which is called “the active space” because it is the only space to be deal with right now and the rest decisions are related with it. In step3, reset $\mathbb{C}'$ through filtering $\mathbb{C}$ by exact restrictions. In step4, blocks made so far by assembling suitable cargos of $\mathbb{C}'$ according to the active space are stored into the set $\mathbb{B}$. In step5, blocks of $\mathbb{B}$ are tried for matching with the active space, and the best block is determined. In step6, an operation of the space split is performed. In step7, all the data stored in $\mathbb{S}$ and $\mathbb{C}$ is updated after a block is determined to be loaded once. When there is no valid space in $\mathbb{S}$ in step2 or no valid block available in step5, the loading procedure is terminated. Summarizing in brief, the procedure mirrors one type of actual container loading and in fact it is the greedy algorithm.

5.3.3 Details of the System Modules

Based on these assumptions and ideas mentioned so far, we designed and implemented the software using a number of programming languages in combination. A

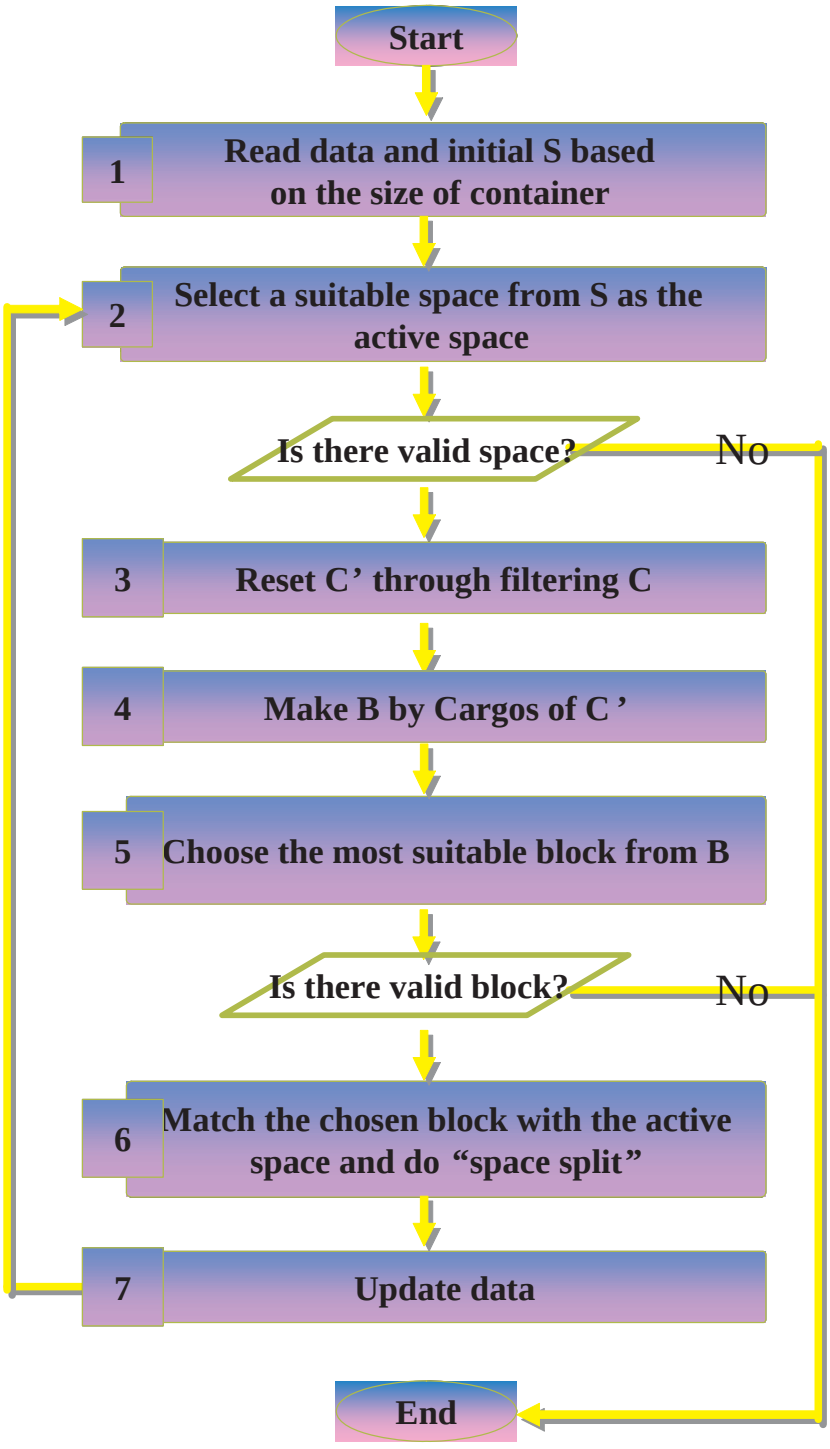


Figure 5.5: Single container loading procedure

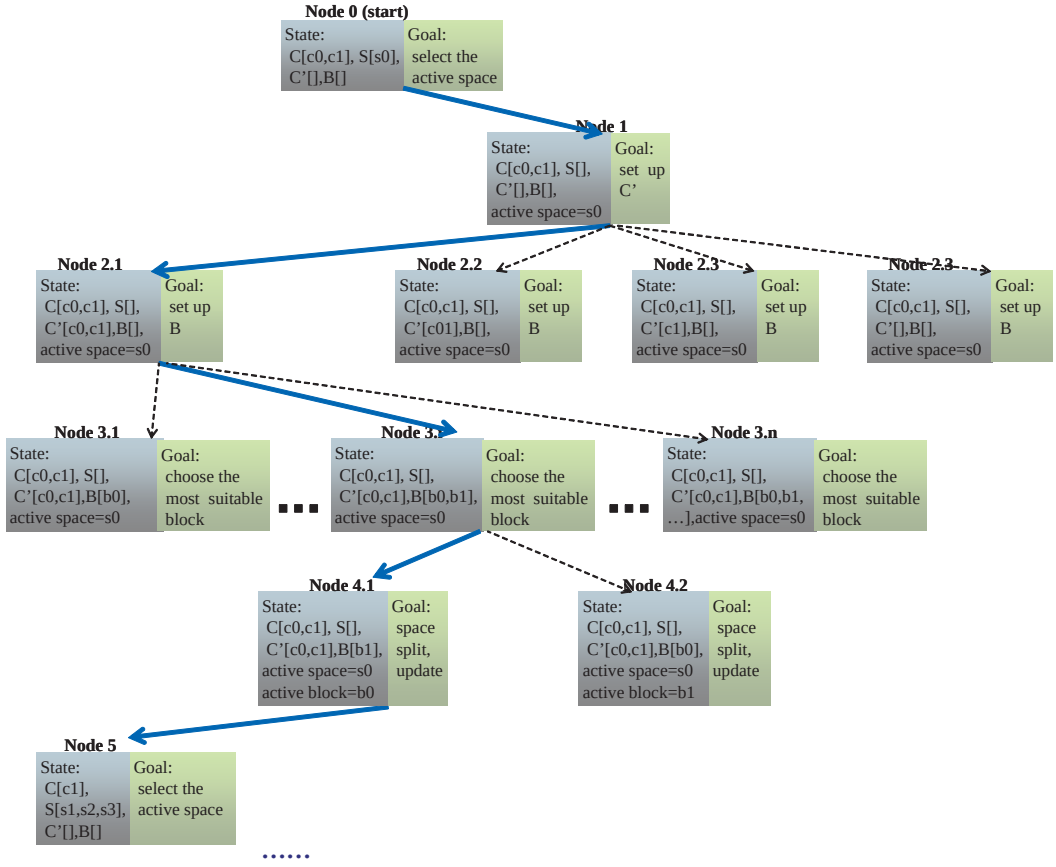


Figure 5.6: Example of the search tree in "Problem Solver"

knowledge part of the system is implemented using PROLOG, and the other parts are by Visual C++. There are five sub modules, "Data Input", "Problem Solver", "Modifying Tool", "Knowledge Base" and "EBL Facility", all of which cooperate well in our software.

A module of "Data Input" reads the basic information on the container and the cargos from a data file (e.g., catalog01.txt).

A module of "Problem Solver" constructs a search tree and identifies the key parts of the tree corresponding to branching decisions such as

1) Which space is to be selected as the active space.

2) How to make blocks using usable cargos.

3) Which block is to be chosen to match with the active space, and so on.

One fragment of a simple running instance in which two kinds of cargos are loaded is presented in Figure 5.6 so as to show how the "Problem Solver" does work. The search process is to extend the leaves of the tree continually until the halt condition is reached.

A module of "Modifying Tool" is used to build an example by the expert through revising the computed result. In the software, it deals with recombining some blocks and changing the locations of the assigned blocks by moving it to more appropriate locations. What's more, the system can analyze the revised result and patterns of blocks and their loading sequences are found. The expert needs to confirm the new result.

A module of "Knowledge Base" stores all fragments of knowledge obtained from the expert. All of the knowledge is described by PROLOG.

5.3.4 System Implementation

To implement the program, first necessary predicates for state descriptions are defined. They are data structures such as *container* (the full information of the loaded container), $usable\_cargos$ (the set $\mathbb{C}$), $usable\_spaces$ (the set $\mathbb{S}$), and so on. Therefore, the state of any node of the search tree can be recorded exactly and the loading process is reflected in the search tree over time.

Second operators and inference rules of the domain knowledge are set up. For instance, the state transition from Node4.1 to Node5 in Figure 5.6 includes the operator "space split" that is defined by a predicate "space split" meaning to divide the active space into new smaller spaces (in this sample, the space "$s0$" is divided and three new spaces "$s1$, $s2$, $s3$" are generated). The sample of the inference rules is defined by a predicate *block_volume* which can compute the volume value of the block from its length, width and height being multiplied in turn. A predicate of *block_volume* plays a key role in the selection among the elements of the set $\mathbb{B}$. Acceptation and/or rejection of the tree extension at Node4.1 and Node4.2 of Figure 5.6 is the result affected by the block volume of the block "$b0$" and the block "$b1$".

Third, control rules are illustrated as follows. There would be many control rules on one branching decision, which prescribes how to select the element among the set. They are grouped into two types "filtering control rules" and "preference control rules". Filtering ones are to decide which element of the set should not be selected, while preference ones must be set to show that this should be executed prior to others because other elements' execution leads to different results. In fact the logical form of a preference control rule is alike "$if..., then...$; $elseif..., then...$; $elseif..., then...$". The selection out of the set is implemented by the filtering control rules as well as the preference ones being executed in turn. The size of the set affected by filtering ones would be reduced. The final answer of the selection operation will be got after preference ones are executed.

Filtering control rules may exist either in *Gcr* or *Scr*. At least one preference control rule with the lowest execution priority must be in *Gcr*, which is the most general selection strategy. Then, it can be guaranteed that the normal motion of the corresponding child decision. Other preference ones are stored either in *Gcr* or *Scr*. For example, the branching decision at Node0 of Figure 5.6 is how to select

the active space from the set $\mathbb{S}$. The corresponding preference control rule is the predicate *select_active_sapce* whose meaning is the space with the smallest start point should be selected from the set $\mathbb{S}$ (the relation $<_p$ is referenced to compare). And a filtering control rule named of *disable_space* means that if the space along the right side of the container is flat it will be filtered out from the set $\mathbb{S}$ is stored in *Scr*.

The final module is "EBL Facility". In order to acquire valid control rules, it also needs to trace and guide the search tree with the assist of "Problem Solver". Note that "Problem Solver" is used not only to solve a new problem but also to analyze an example using knowledge stored within "Knowledge base". When solving a new problem, it only uses a stable set of "control rules" of "Knowledge base". When analyzing an example, it has to consider how to set up a stable set of "control rules" of "Knowledge base". If the learned control rule belongs to "filtering control rules", its executing priority would be calculated and the "if/then/else" structure described in the preceding paragraph is maintained. Furthermore, this module has necessary functions to check the existence of any exceptions and functions as a friendly and interactive interface for the expert.

5.4 Experiments

Some fragments of real examples output by the system are presented in Figure 5.7 and Figure 5.8.

5.4.1 Experiment I

This sample is to generalize tact knowledge about the branching decision on how to select an active space from whole usable spaces.

In Figure 5.7, four loading results shown on the left are generated by "Problem Solver" using only *Gcr*. They can be thought as a subset of solutions that are commonly sharing unsatisfactory aspects, because they have the same disadvantage on utilizing the right flat space. Tacit knowledge provided by the expert to exclude these disadvantage is that if the right flat space is only available to load a few cargos it should not be used because it is not easily managed during loading and unloading processes.

The result provided by the expert using "Modifying Tool" is shown in the right-top of Figure 5.7. Then, this result is regarded as a successful example and is to be analyzed by "EBL Facility". In our software, the knowledge mentioned in the above is the "filtering" one and has been additionally stored in *Scr*.

At last "EBL Facility" forms an explanation on why this example is satisfactory and this knowledge is confirmed to be valid to the current subset as mentioned at the end of the previous chapter. What is more, a control rule with more vigorous and

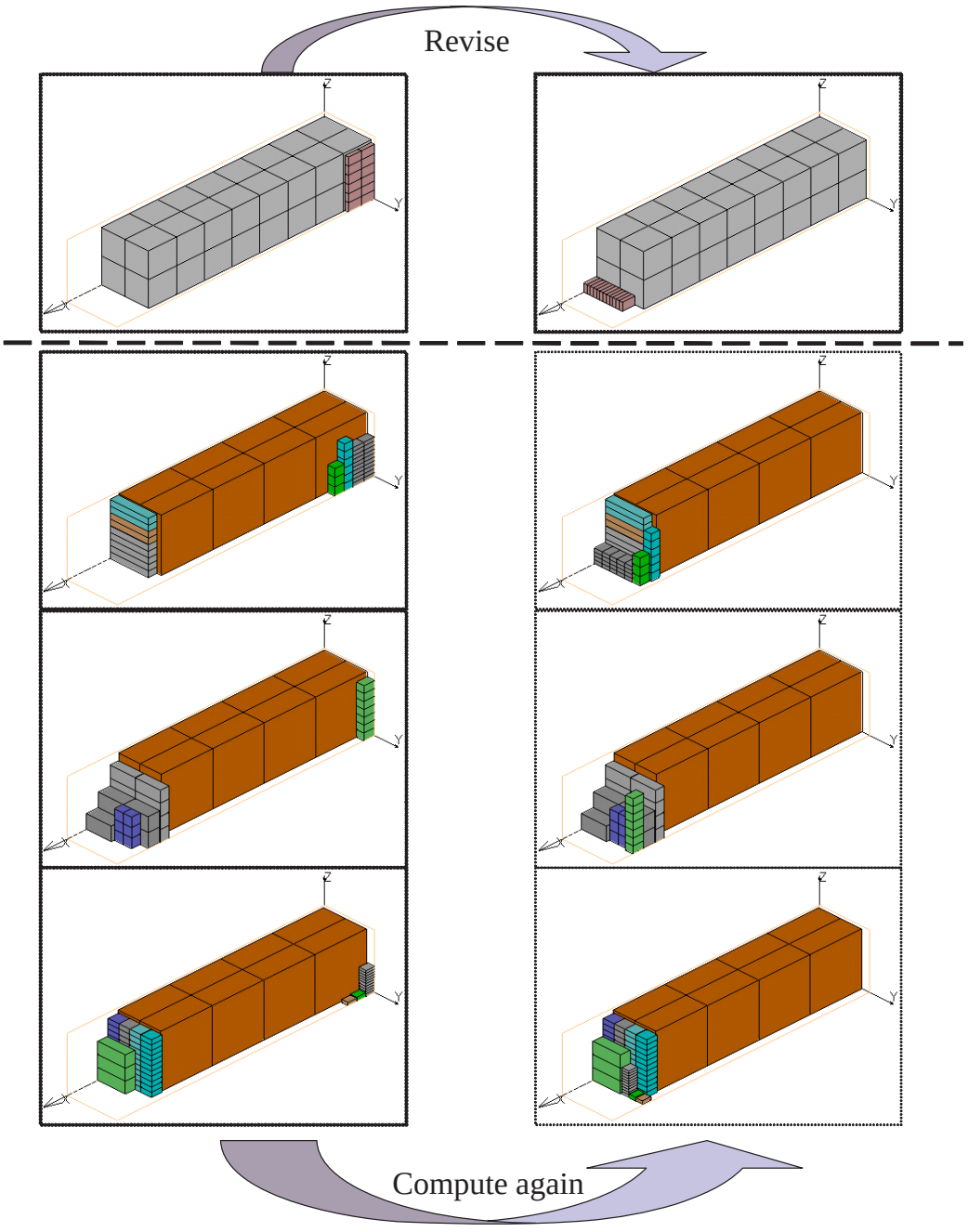


Figure 5.7: Example One for capturing knowledge

succinct formation is added into *Rcr'* because the regressing of the explanation tree can find the weakest conditions under which the proof will hold. The real predicate expressed in PROLOG is:

$$disable\_space(Spc) :- space(position(\_, Posys, \_), size(\_, Ws, \_)), \\ container(size(\_, Wc, \_), \_), (Posys + Ws) =:= Wc, Ws =< 450. \tag{5.1}$$

In the expression 5.1 "space" and "container" are predicates of data structures and variables including upper characters are affiliated with these two data structures. Here a constant 450 is the experiential value defined by the expert and points out that only the space whose width value is less or equal 450 is regarded as the flat space.

After getting *Rcr'*, the problem solver calculates others of the subset again and it should use not only *Gcr* but also *Rcr'*. The final results of the subset are the right ones of Figure 5.7, in which all cargos are easier for the warehouse operators to manage.

5.4.2 Experiment II

The generalized tacit knowledge about the branching decision on how to select the most suitable block from the set $\mathbb{B}$ that matches with the active space is illustrated as follows.

Concerning with the branching decision of the block selection, the default control rule is that the block with the largest volume should be chosen. However, there is another preference control rule stored in *Scr*: if the space is a strip of the container (i.e., the width and the height of the space are equal with the width and the height of the container, respectively) and there is at least one block whose Y-Z area ratio (i.e. Y-Z surface area of the block is divided by Y-Z surface area of the container) is not less than 95% (experiential value), the block with the largest Y-Z area ratio should be picked up from the set $\mathbb{B}$. In this case, the loading can become more stable.

The initial control rule in *Scr* is:

$$select\_block(UsblBlcks, ABlck) :- active\_space(Spc), strip\_space(Spc), \\ yz\_biggest\_block(UsblBlcks, ABlck), get\_block\_yz(ABlck, YZArea1), \\ get\_space\_yz(Spc, YZArea2), (YZArea1/YZArea2) >= 0.95. \tag{5.2}$$

In expression 5.2, a predicate of *active_space* is for the data structure of the active space, and a predicate of *strip_space* is used to judge whether one space is the strip of the container or not. A predicate of *yz_bigest_block* is an atomic inference rule for finding out the block with largest Y-Z area among the set $\mathbb{B}$. Predicates of *get_blcok_yz* and *get_space_yz* are used to calculate the Y-Z area of the block and the space, respectively.

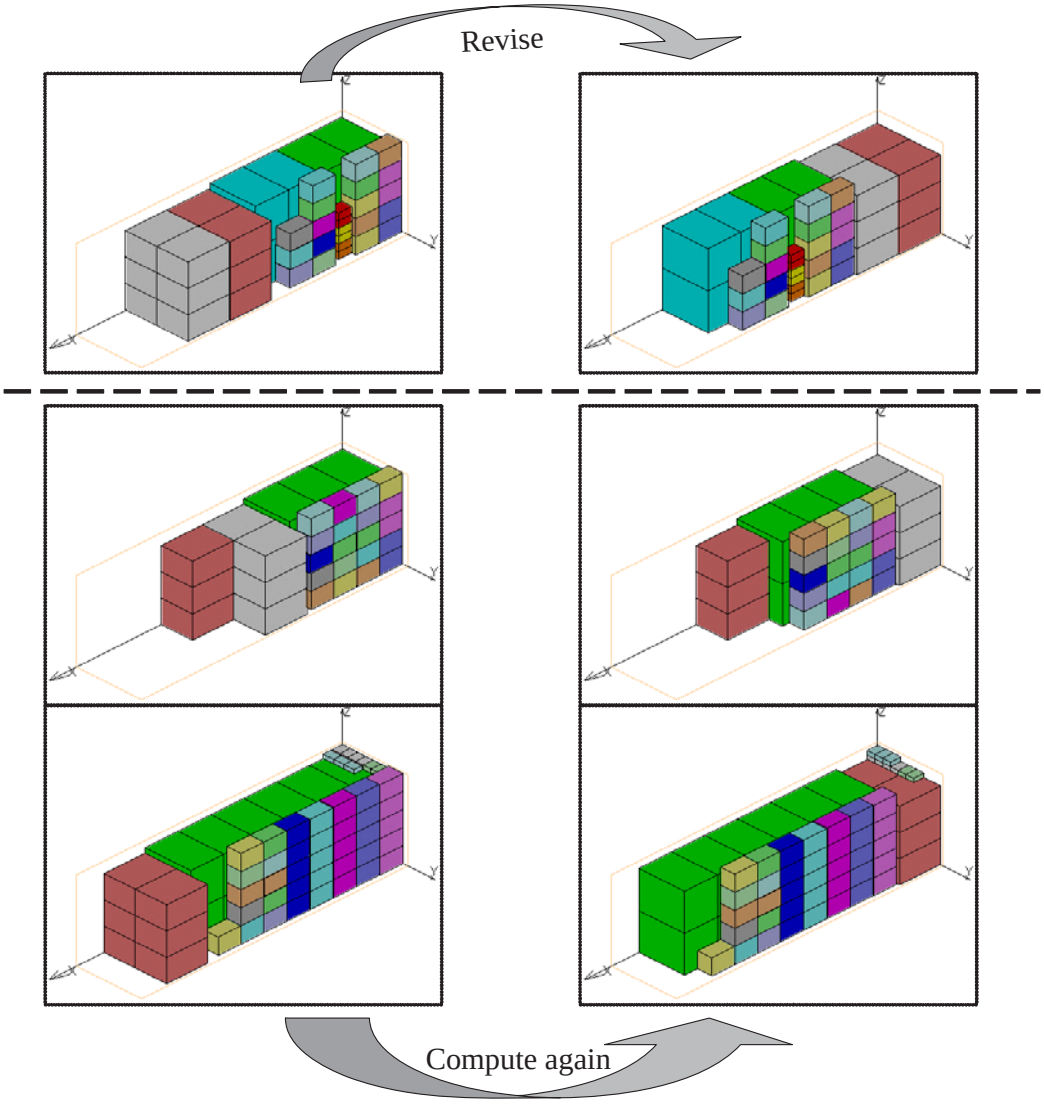


Figure 5.8: Example Two for capturing knowledge

Finally the learned control rule stored in *Rcr'* is:

$$
\begin{aligned}
&select\_block(UsblBlcks, ABlck) :- \\
&\quad active\_space(space(pos(\_, Pys, Pzs), size(\_, Ws, Hs))), \\
&\quad Pys =:= 0, Pzs =:= 0, container(size(\_, Wc, Hc), \_), \\
&\quad Ws =:= Wc, Hs =:= Hc, yz\_biggest\_block(UsblBlcks, ABlck), \\
&\quad ABlck = block(\_, \_, sub\_size(\_, SWb, SHb), \_, \_, num(\_, Nw, Nh)), \\
&\quad SWb * Nw * SHb * Nh)/(Wc * Hc)) >= 0.95.
\end{aligned}
\tag{5.3}
$$

5.5 Conclusion

In this research, we presented some experiments of container loading using our IKAF, and showed that this is a feasible methodology to learn the expert's tacit knowledge through analyzing and explaining the expert's providing examples. The tacit knowledge of the expert on how to use his/her knowledge (i.e., a meta-knowledge) was captured by organizing and prioritizing control rules through auto analysis and valid verification. We showed advantages realized by combining the general heuristic algorithm with the expert's knowledge. It should be noticed that IKAF provides a platform for the expert to express his/her experiential knowledge. In addition, the system is open and the method to solve one hard problem is dynamic. This is the biggest difference between IKAF and other traditional algorithms.

Chapter 6

Conclusion

This research discusses how to solve the *constraint satisfaction problem* in the real world using approaches based on distributed and cooperative systems. Three approaches are proposed: TOCM, TZBM and IKAF in the order of low-flexibility to high-flexibility.

In the optimization approach based on distributed and cooperative systems, an original problem is solved by decomposing it into multiple agents. Proposed approaches TOCM and TZBM can solve complicated problems in a flexible mode because they use a relaxation mechanism and highly-efficient negotiation strategy. TOCM can classify restrictions and prioritize them. It improves a solution by eliminating the constraint agent of the system. Further, TZBM is used to eliminate all violations on restrictions to the greatest extent possible. Put simply, the proposed approaches do not seek an optimal solution, but find a satisficing solution within a limited computation time. One complex loading pattern of the container loading problem, the *two-row pattern*, is discussed and solved by the proposed approaches. Results show that the approaches succeed.

In the machine learning approach based on distributed and cooperative systems, a human expert agent is embedded in a solving process to further improve flexibility and adaptability. In the proposed IKAF approach, a human expert can improve the solving quality by criticizing an initial solution computed by a computer agent. This is possible because a human expert has intelligence to recognize complicated situations and then to employ the best solving strategy. In practice, IKAF is an available platform to augment the expert's tacit knowledge. Therefore, knowledge obtained from a human expert agent can be shared with other human agents, especially unskilled human agents. The approach is validated by solving the container learning problem, which includes extreme heterogeneous cargoes. IKAF is still immature and future research is needed to strengthen it.

Bibliography

[1] Bischoff, E. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, Vol. 168, No. 3, pp. 952–966, 2006.

[2] Bischoff, E. and M. Ratcliff. Issues in the development of approaches to container loading. *Omega*, Vol. 23, No. 4, pp. 377–390, 1995.

[3] Boddy, M. and T. Dean. An analysis of time-dependent planning. *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pp. 49–54, 1988.

[4] Bortfeldt, A. and H. Gehring. Applying tabu search to container loading problems. *Fern Universitat Hagen Technical Report*, 1998.

[5] Bortfeldt, A., H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, Vol. 29, No. 5, pp. 641–662, 2003.

[6] Davies, A. and E. Bischoff. Weight distribution considerations in container loading. *European Journal of Operational Research*, Vol. 114, No. 3, pp. 509–527, 1999.

[7] Dyckhoff, H. A topology of cutting and packing problems. *European Journal of Operational Research*, Vol. 44, No. 2, pp. 145–159, 1990.

[8] Eley, M. Solving container loading problems by block arrangement. *European Journal of Operational Research*, Vol. 141, No. 2, pp. 393–409, 2002.

[9] Ellman, T. Explanation-based learning: a survey of programs and perspectives. *ACM Computing Surveys (CSUR)*, Vol. 21, No. 2, pp. 163–221, 1989.

[10] Faroe, O., D. Pisinger, and M. Zachariasen. Guided Local Search for the Three-Dimensional Bin-Packing Problem. *INFORMS Journal on Computing*, Vol. 15, No. 3, pp. 267–283, 2003.

[11] Gehring, H. A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, Vol. 4, No. 5-6, pp. 401–418, 1997.

[12] Gehring, H. and A. Bortfeldt. A Parallel Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research*, Vol. 9, No. 4, pp. 497–511, 2002.

[13] Goldratt, E. and J. Cox. *The Goal: a process of ongoing improvement*. The North River Press, 1992.

[14] He, D. and A. Babayan. Agent-based agile manufacturing system scheduling. *International Conference on Industrial Engineering and Production Management*, 2001.

[15] Hifi, M. Approximate algorithms for the container loading problem. *International Transactions in Operational Research*, Vol. 9, No. 6, pp. 747–774, 2002.

[16] Horvitz, E. Reasoning about beliefs and actions under computational resource constraints. *Uncertainty in Artificial Intelligence*, Vol. 3, pp. 301–324, 1989.

[17] Kalczynski, P. and L. Dabrowski. Multi Container Loading With Grouping Goals: An Agent-Based Approach. pp. 71–76, April 2004.

[18] Leonard-Barton, D. and S. Sensiper. The role of tacit knowledge in group innovation. *California Management Review*, Vol. 40, No. 3, pp. 112–132, 1998.

[19] Lodi, A., S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, Vol. 141, No. 2, pp. 241–252, 2002.

[20] Mack, D., A. Bortfeldt, and H. Gehring. A parallel hybrid local search algorithm for the container loading problem. *International Transactions in Operational Research*, Vol. 11, No. 5, pp. 511–533, 2004.

[21] Michael, P. The Tacit Dimension. *New York: Doubleday&Company Inc*, pp. 29–52, 1966.

[22] Minton, S., J. Carbonell, C. Knoblock, D. Kuokka, and O. Etzioni. Explanation-based learning: a problem solving perspective. *Artificial Intelligence*, Vol. 40, No. 1, pp. 63–118, 1989.

[23] Mitchell, C. Machine Learning, 1997.

[24] Morton, T. and D. Pentico. *Heuristic Scheduling System*. Wiley Inter science, 1993.

[25] Pisinger, D. Heuristics for the container loading problem. *European Journal of Operational Research*, Vol. 141, No. 2, pp. 382–392, 2002.

[26] Russell, S. and P. Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1995.

[27] Scheithauer, G. Algorithms for the container loading problem. *Operational Research Proceedings 1991*, pp. 445–452, 1992.

[28] Simon, H. *The Sciences of the Artificial*. MIT Press, 1981.

[29] Smith, D. *The Measurement Nightmare: How the Theory of Constraints Can Resolve Conflicting Strategies, Policies and Measures*. St. Lucie Press, 2000.

[30] Takahara, S. Loading Problem in Multiple Containers and Pallets Using Strategic Search Method. *Modeling Decisions for Artificial Intelligence: Second International Conference, MDAI 2005, Tsukuba, Japan, July 25-27, 2005: Proceedings*, 2005.

[31] Terno, J., G. Scheithauer, U. Sommerweiß, and J. Riehme. An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research*, Vol. 123, No. 2, pp. 372–381, 2000.

[32] Tian, Y., K. Shimohara, Y. Liu, and T. Sawaragi. Cooperative negotiation strategy in multi-agent system. *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, Vol. 3, , 2003.

[33] Tsang, E. Foundations of constraint satisfaction. Computation in Cognitive Science, 1993.

[34] Waltz, D. Understanding line drawings of scenes with shadows. *The Psychology of Computer Vision*, pp. 19–91, 1975.

[35] Xue, J. and K. Lai. Effective Methods for a Container Packing Operation. *Mathematical and Computer Modelling*, Vol. 25, No. 2, pp. 75–84, 1997.

[36] Yokoo, M. *Distributed constraint satisfaction: foundations of cooperation in multi-agent systems*. Springer-Verlag, London, UK, 2001.

Published Papers

Journal Papers

1. Y. Liu, Y. Tian, T. Sawaragi, "A TOC-based heuristic algorithm for solving two-row pattern container loading problem", *International Journal of Services Operations and Informatics* , Vol. 2, No. 4, pp. 339–356, November 2007.

2. Y. Liu, Y. Tian and T. Sawaragi, "Applying DBR and MAS to Solving Container Loading Problem", *Journal of Japan Society for Fuzzy Theory and Intelligent Informatics*, Vol. 18, No. 6, pp. 837–848, December 2006.

3. T. Sawaragi, Y. Liu, Y. Tian, "Human-Machine Collaborative Knowledge Creation: Capturing Tacit Knowledge by Observing Expert's Demonstration of Load Allocation", *International Journal of Knowledge and Systems Sciences*, Vol. 3, No. 2, pp. 9–19, June 2006.

International Conference Papers

4. Y. Liu, Y. Tian and T. Sawaragi, "A Heuristic Algorithm Based on DBR and MAS for Solving Container Loading Problem", *Proc. of the 2006 IEEE International Conference on Service Operations and Logistics, and Informatics*, pp. 476–481, 2006.

5. Y. Liu, T. Sawaragi and Y. Tian, "Learning Generalized Tacit Knowledge by Explaining Experts' Performance of Load Allocation Process", *Proc. of the SICE Annual Conference 2005*, pp. 3248–3253, 2005.

6. Y. Tian, Y. Liu, N. Sannomiya and K. Shimohara, "Three-zone Buffer Management and Its Applications to Equalizing Operations Problems", *Proc. of International Manufacturing & Logistics Systems, Technical Session III-1*, pp. 1–6, 2005.

7. T. Sawaragi, Y. Liu, Y. Tian, "Human-Machine Collaborative Knowledge Creation: Capturing Tacit Knowledge by Observing Expert's Demonstration of

Load Allocation", *Proc. of CSM/KSS 2005 on the web*, 2005.

8. Y. Tian, K. Shimohara, T. Sawaragi and Y. Liu, "An Autonomous and Cooperative Multi-agent Model for Solving Real World Problems", *Proc. of the 7th World Multi-Conference on Systemics, Cybernetics and Informatics*, pp. 272–277, 2003.

9. Y. Tian, Y. Liu, K. Shimohara and T. Sawaragi, "Cooperative Negotiation Strategy in Multi-agent System", *Proc. of the 42nd IEEE Conference on Decision and Control*, pp. 2549–2554, 2003.

Japanese Conference Papers

10. P. Hui, Y. Tian, Y. Liu, T. Sawaragi, "TZBM Approach for Solving Hybrid Flow Shop Scheduling Problem", **第 *20* 回自律分散システム・シンポジュウム**, pp. 35–40, 2008.

11. Y. Liu, T. Sawaragi, Y. Tian, "Application of multi-agent system for solving container loading problem", **第 *19* 回自律分散システム・シンポジュウム**, pp. 359–364, 2007.

12. 田, 劉, 三宮, 下原, "人間-機械インタラクションにおける共有領域知識ベースの構築", **第 *18* 回自律分散システム・シンポジウム資料**, pp. 43–48, 2006.

13. 田, 劉, 椹木, 下原, "積載問題における人間・機械インタラクション", **第 *32* 回知能システム・シンポジウム資料**, pp. 145–150, 2005.

14. 田, 劉, 下原, 椹木, "TZBM アプローチを用いた作業の平準化モデル", **第 *3* 回 *FIT2004* 講演論文集**, pp. 15–19, 2004.

15. 田, 下原, 劉, 椹木, "スリーゾーンのバッファ・マネジメント・アプローチ", **第 *16* 回自律分散システム・シンポジウム資料**, pp. 171–176, 2004.

16. 田, 椹木, 劉, "配車配送問題の DBR に基づく定式化解評価および意思決定手法", **第 *29* 回知能システム・シンポジウム資料**, pp. 41–46, 2002.

17. 田, 椹木, 劉, "制約緩和の社会的交渉モデルに基づく積載スケジューリング・マルチ・エージェントシステム", **電気学会 電子・情報・システム部門 *(C* 部門*)* 大会資料**, pp. 348–353, 2002.

Patent

18. 田 雅杰，下原 勝憲，椹木 哲夫，劉 遠, “特願 2003-016134(管理装置，管理方法および管理プログラム): Drum-Buffer-Rope(DBR) をアプローチ化した Three-zone Buffer Management(TZBM) に関する特許である”, **国際電気通信基礎技術研究所**, 2004.